НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

кафедра автоматика та управління в технічних системах
(повна назва кафедри)

«На правах рукопису»
УДК _____

«До захисту допущено»
Завідувач кафедри
_____ О. І. РОЛІК
(підпис) (ініціали, прізвище)
“___”_____2018 р.

Магістерська дисертація

зі спеціальності (спеціалізації) 126 «Інформаційні системи та технології»
(код і назва спеціальності)

на тему: Система автоматизації процесів тестування програмного забезпечення з використанням паралелізації тестів

Виконав : студент __6_ курсу, групи ____ІА–72мп_____
(шифр групи)

Бєй Олександр Вікторович _____ _____
(прізвище, ім’я, по батькові) (підпис)

Науковий керівник Чемерис О.А. _____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант_____ _____ _____
(назва розділу) (науковий ступінь, вчене звання, , прізвище, ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2018 року

Національний технічний університет України

“Київський політехнічний інститут

імені Ігоря Сікорського”

Факультет інформатики та обчислювальної техніки

(повна назва)

Кафедра автоматики та управління в технічних системах

(повна назва)

Ступінь вищої освіти –другий (магістерський)

(код, назва)

Спеціальність 126 «Інформаційні системи та технології»

(код, назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ О. І. РОЛІК

(підпис) (ініціали, прізвище)

“___”_____2018_р.

ЗАВДАННЯ

на магістерську дисертацію студенту

Бєю Олександру Вікторовичу

(прізвище, ім’я, по батькові)

1.**Тема дисертації**__ Система автоматизації процесів тестування програмного забезпечення з використанням паралелізації тестів

_____

Науковий керівник дисертації__ Чемерис Олександр Анатолійович, к. т. н., доцент_____

затверджені наказом по університету від “_29_” _жовтня_2018_р.№_____

2.Строк подання студентом дисертації “_4_” грудня 2018_р.

3. Об`єкт дослідження: експорт Save As функціонал сайту PDFfiller

4.Зміст пояснювальної записки**:** а) призначення та область застосування; б) Визначення тестування; в) огляд існуючих рішень; г) Вибір технології для реалізації системи; д) Реалізація системи; е) Тестування системи

5. Перелік графічного (ілюстративного) матеріалу

Структурна схема системи, Діаграма варіантів використання, Діаграма послідовності, Діаграма класів фреймворка для автоматизації, Структура проекту у інтегрованому середовищі розробки, Алгоритм роботи системи, Алгоритм роботи виявлення помилок при тесті, Життєвий цикл програмного забезпечення з використанням розробленої системи

6. Консультанти розділів проекту:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання “_29_” __жовтня__2018_р.

Календарний план

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів проекту	Примітка
1	Огляд існуючих рішень та розробка структури системи	10.10.2018	
2	Вибір технології для розробки	15.10.2018	
3	Вибір інструментів для тестування	20.10.2018	
4	Розробка схеми алгоритму роботи	30.11.2018	
5	Написання тест кейсів	5.11.2018	
6	Написання фреймворку для тестування	10.11.2018	
7	Написання автоматизованих тестів	20.11.2018	
8	Тестування системи	01.12.2018	
	Представлення до захисту	4.12.2018	

Студент _____ __ Бєй О.В.

(підпис) (ініціали, прізвище)

Керівник проекту _____ _ Чемерис О.А.

АНОТАЦІЯ

У магістерській дисертації розроблено систему автоматизації процесів тестування програмного забезпечення із використанням паралелізації тестів.

Метою даної роботи було створення системи для автоматичної перевірки функціоналу тестованого об'єкта зі зручним інтерфейсом виводу помилок. Відповідно до поставленої мети було розроблено та протестовано систему автоматизації процесів тестування програмного забезпечення із використанням паралелізації тестів. В дисертації розроблено структурна схема, схема послідовності, діаграма варіантів використання, алгоритм роботи системи та інші.

В роботі розглянуто особливості деяких інструментів для автоматизації тестування та успішно використанні на практиці.

Значну увагу в роботі приділено розробці фреймворка для тестування, завдяки якому швидкість написання автоматизованих тестів значно зростає. При цьому було розроблено схему роботи системи та робочу програму мовою Java. Результати цього моделювання підтвердили достовірність теоретичних відомостей.

Робота може бути корисною для компаній, націлених на розвиток напрямку тестування, автоматизації процесу тестування і підвищення якості свого продукту, оскільки в роботі описані необхідні теоретичні основи, більш того, проаналізовані критерії ефективності процесу тестування.

Робота містить 100 с. тексту, 49 рисунків, 20 літературних джерел та 3 додатка.

Ключові слова: система автоматизованого тестування, якість програмного забезпечення, тестувальник, паралелізація тестів, запуск тестів автоматизоване тестування, результати тесту, тестування, помилки.

SUMMARY

In the master's thesis was developed the system of software testing processes automation with the use of parallelization of tests.

The purpose of this work was to create a system for automatically verifying the functionality of the tested object with a user-friendly error-handling interface. In accordance with the stated goal, the system of automation of software testing processes was developed and tested with the use of parallel tests. In the dissertation the structural scheme, the sequence diagram, the diagram of variants of use, the algorithm of system operation and others are developed.

The paper considers the features of some tools for testing automation and successfully used in practice.

Considerable attention in the work is devoted to the development of a framework for testing, thanks to which the speed of writing automated tests greatly increases. At the same time, the scheme of work of the system and the working program in the Java language was developed. The results of this simulation confirmed the reliability of the theoretical information.

The work may be useful for companies aimed at developing the testing direction, automating the testing process and improving the quality of their product, since the paper describes the necessary theoretical foundations, moreover, analyzes the criteria for the effectiveness of the testing process.

The work contains 100 s. text, 49 drawings, 20 literary sources and 3 appendices.

Key words: automated testing system, software quality, testator, test parallelization, automated test run tests, test results, testing, errors.

Зміст

ВСТУП 11
1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ 13
2 ВИЗНАЧЕННЯ ТЕСТУВАННЯ 13
2.1 Що таке тестування ПО 13
2.2 Життєвий цикл продукту та тестування 14
2.3 Місце тестування в процесі розробки ПО 17
2.4 Методологія тестування 19
2.5 Рівні тестування програмного забезпечення 21
2.6 Види тестування 24
Висновки з розділу 29
3 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ 30
3.1 Проблеми тестування програмного забезпечення 30
3.2 Katalon Studio 32
3.3 Selenium IDE 34
Висновки з розділу 36
4 ВИБІР ТЕХНОЛОГІЇ ДЛЯ РЕАЛІЗАЦІЇ СИСТЕМИ 37
4.1 Вибір мови програмування 37
4.2 Вибір інтегрованого середовища розробки 43
4.3 Selenium 47
4.4 TestNG 51
4.5 Jenkins 54
Висновки з розділу 55
5 РЕАЛІЗАЦІЯ СИСТЕМИ 56
5.1 Опис системи 56
5.2 Вибір об’єкту для тестування 57
5.2 Вибір оптимальної кількості тестів для перевірки вибраної частини продукту 63
5.4 Написання тест кейсів 63
5.5 Написання фреймворка для тестування 72
5.6 Написання автоматизованих тестів 78
5.7 Паралелізація тестів використовуючи фреймворк TestNG 81
5.8 Налаштування та запуск тестів 82

Висновки з розділу 88
6 ТЕСТУВАННЯ СИСТЕМИ 90
6.1 Прогін тестів та аналіз 90
6.2 Тест з помилкою 93
Висновки з розділу 93
7 РОЗРОБЛЕННЯ СТАРТАП-ПРОЕКТУ 94
7.1. Опис ідеї проекту (товару, послуги, технології) 94
7.2. Технологічний аудит ідеї проекту 96
7.3. Аналіз ринкових можливостей запуску стартап-проекту 96
7.4. Розроблення ринкової стратегії проекту 104
7.5. Розроблення маркетингової програми стартап-проекту 106
Висновки з розділу 109
ВИСНОВКИ ДО МАГІСТЕРЬСКОЇ ДИСЕРТАЦІЇ 109
СПИСОК ПОСИЛАНЬ 111

ПЕРЕЛІК СКОРОЧЕНЬ

ПО - Програмне забезпечення

RUP - Rational Unified Process - раціональний уніфікований процес розробки програмного забезпечення, створений компанією Rational Software

Баг - У програмуванні жаргонне слово, зазвичай позначає помилку в програмі або системі, яка видає несподіваний або неправильний результат

MDAC - Microsoft Data Access Components - сукупність технологій компанії Microsoft, що дозволяють отримати уніфікований спосіб доступу до даних з різних реляційних і не реляційних джерел. Термін MDAC є загальним позначенням для всіх розроблених компанією Microsoft технологій, пов'язаних з базами даних

ООП - Об'єктно-орієнтоване програмування

ОПФ - Організаційно-правова форма

FTP File Transfer Protocol - стандартний протокол, призначений для передачі файлів по TCP-мереж

JDBC Java DataBase Connectivity - переносних незалежний промисловий стандарт взаємодії Java-додатків з різними СУБД

СУБД - Система управління базами даних

ОПФ - Організаційно-правова форма

ОВС - Основний вид діяльності

GUI - Різновид призначеного для користувача інтерфейсу, в якому елементи інтерфейсу (меню, кнопки, значки, списки і т. п.), представлені користувачеві на дисплеї, виконані у вигляді графічних зображень

ІС - Інформаційна система

ІТ - Інформаційні технології

ВСТУП

На даний момент в сучасному світі все дуже швидко розвивається. Майже в кожній сфері бізнесу зараз повинен бути свій сайт. Над цим сайтом трудиться деяка група людей. Іноді це велика кількість, іноді менше. Все залежить від того, наскільки сайт важливий для бізнесу.

Існують різні великі продукти, такі як Google, Facebook, Twitter і так далі, де кількість відвідувачів безпосередньо залежить від якості веб додатку. Зовсім недавно стала відомою робота тестувальника. Тестувальник - невід'ємна частина робочого процесу життєвого циклу програмного забезпечення, людина, яка відповідає за якість продукту, що випускається. Існує ручний тестувальник і автоматизований тестувальник. Перший перевіряє тестований об'єкт за допомогою різноманітних технік, пише документацію, баг репорти, стежить за процесом виправлення помилок. Завдання автоматизованого тестувальника - написати програми, які будуть тестувати певний функціонал тестованого об'єкта.

Кожна група розробників програмного забезпечення тестує свої продукти, проте програмне забезпечення завжди має якісь помилки. Ручні тестувальники прагнуть зловити їх до випуску продукту, але вони завжди закрадаються і часто з'являються знову, навіть при найкращих ручних процесах тестування. Впровадження процесу автоматизованого тестування - кращий спосіб підвищити ефективність і охоплення тестування програмного забезпечення.

Ручне тестування програмного забезпечення виконується людиною, що сидить перед комп'ютером, ретельно переглядає екрани додатків, пробуючи різні комбінації використання і введення. Потім порівнюють результати з очікуваною поведінкою і записуючи свої спостереження. Ручні тести часто повторюються протягом циклів розробки для змін вихідного коду та інших ситуацій, таких як множинні операційні середовища і конфігурації обладнання.

Інструмент автоматичного тестування здатний відтворювати попередньо записані і попередньо визначені дії, порівнювати результати з очікуваною

поведінкою і повідомляти про успіх чи невдачу цих ручних тестів інженеру по тестуванню. Після створення автоматичних тестів їх можна легко повторювати і розширювати для виконання завдань, неможливих при ручному тестуванні. Через це досвідчені менеджери виявили, що автоматизоване тестування програмного забезпечення є важливим компонентом успішних проектів.

Автоматизоване тестування програмного забезпечення довгий час вважалося критично важливим для великих організацій, що займаються розробкою програмного забезпечення, але часто вважається занадто дорогим або важким для впровадження невеликими компаніями.

Тести програмного забезпечення повинні часто повторюватися під час циклів розробки, щоб гарантувати якість. Кожен раз, коли вихідний код змінюється, тести програмного забезпечення повинні повторюватися. Для кожного випуску програмного забезпечення воно може бути протестовано на всіх підтримуваних операційних системах і конфігураціях обладнання. Повторення цих тестів вручну вимагає великих витрат часу і коштів. Після створення автоматизовані тести можна запускати знову і знову без додаткових витрат, і вони набагато швидше, ніж ручні тести. Автоматизоване тестування програмного забезпечення може скоротити час виконання повторюваних тестів з кількох днів до кількох годин. Економія часу, яка безпосередньо призводить до економії грошей.

Автоматичне тестування програмного забезпечення може збільшити глибину і обсяг тестів, щоб допомогти поліпшити якість програмного забезпечення. Тривалі тести, яких часто уникають при ручному тестуванні, можна запускати без нагляду. Їх навіть можна запускати на декількох комп'ютерах з різними конфігураціями. Автоматичне тестування програмного забезпечення може заглянути всередину додатку і побачити вміст пам'яті, таблиці даних, вміст файлу і внутрішні стану програми, щоб визначити, чи веде продукт себе так, як очікувалося. Автоматизація тестування може легко виконати тисячі різних складних тестових випадків під час кожного запуску тесту, забезпечуючи покриття, яке неможливо при ручних тестах.

У рамках магістерської дисертації маємо за мету спроектувати систему, яка дасть можливість автоматично перевіряти на правильність тестований об’єкт та отримувати звіт по результату. Головною ціллю є ефективне використання автоматизованого тестування для перевірки якості ПО.

Під час створення програмного продукту має бути проведений повний цикл його написання: починаючи від постановки завдання, вимог до продукту, до безпосередньо написання програмної частини, її тестування та створення відповідної технічної документації для більшого розуміння роботи.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Система автоматизації процесів тестування програмного забезпечення з використанням паралелізації тестів актуальна для компаній, націлених на розвиток напрямку тестування, автоматизації процесу тестування і підвищення якості свого продукту, оскільки в роботі описані необхідні теоретичні основи, більш того, проаналізовані критерії ефективності процесу тестування і на конкретному прикладі розрахована ефективність впровадження автоматизованого тестування в компанії.

Головне завдання системи – проаналізувати ефективність використання автоматизованого тестування для перевірки якості програмного забезпечення. Система показує, як можна заощадити час на тестування використовуючи автоматизовані тести та їх паралельний запуск.

Основною складовою системи є автоматизовані тести, які можуть перевіряти функціональність системи, правильність відображення графічного інтерфейсу.

2 ВИЗНАЧЕННЯ ТЕСТУВАННЯ

2.1 Що таке тестування ПО

Тестування - це процес аналізу ПО, спрямований на виявлення відмінностей між його реально існуючими і необхідними властивостями (дефект) і на оцінку властивостей ПО [5].

В життєвому циклі ПЗ визначені серед інших допоміжні процеси верифікації, атестації, спільного аналізу та аудиту. Процес верифікації є процесом визначення того, що програмні продукти функціонують в повній відповідності з вимогами або умовами, реалізованими в попередніх роботах. Даний процес може включати аналіз, перевірку та випробування (тестування). Процес атестації є процесом визначення повноти відповідності встановлених вимог, створеної системи або програмного продукту їх функціональному призначенню. Процес спільного аналізу є процесом оцінки станів і, при необхідності, результатів робіт (продуктів) за проектом. Процес аудиту є процесом визначення відповідності вимогам, планам та умовам договору. У сумі ці процеси і складають те, що зазвичай називають тестуванням.

Тестування ґрунтується на тестових процедурах з конкретними вхідними даними, початковими умовами і очікуваним результатом, розробленими для певної мети, такої, як перевірка окремої програми або верифікація відповідності на певну вимогу.

Тестові процедури можуть перевіряти різні аспекти функціонування програми від правильної роботи окремої функції до адекватного виконання бізнес-вимог.

При виконанні проекту необхідно враховувати, відповідно до яких стандартів і вимогам буде проводитися тестування продукту. Які інструментальні засоби будуть (якщо будуть) використовуватися для пошуку і для документування знайдених дефектів. Якщо пам'ятати про тестування з самого початку виконання проекту, тестування продукту не завдасть неприємних несподіванок. А значить і якість продукту, швидше за все, буде досить високим.

2.2 Життєвий цикл продукту та тестування

Все частіше використовуються ітеративні процеси розробки ПЗ, зокрема, технологія RUP - Rational Unified Process. На рисунку 2.1 можна побачити життєвий цикл продукту по RUP. При використанні такого підходу тестування перестає бути процесом «на відшибі», який запускається після того, як

програмісти написали весь необхідний код. Робота над тестами починається з самого початкового етапу виявлення вимог до майбутнього продукту і тісно інтегрується з поточними завданнями. І це висуває нові вимоги до тестувальникам. Їх роль не зводиться просто до виявлення помилок у можливій повноті і якомога раніше. Вони повинні брати участь в загальному процесі виявлення та усунення найбільш істотних ризиків проекту. Для цього на кожну ітерацію визначається мета тестування і методи її досягнення. А наприкінці кожної ітерації визначається, наскільки ця мета досягнута, чи потрібні додаткові випробування, і чи не потрібно змінити принципи та інструменти проведення тестів. В свою чергу, кожен виявлений дефект повинен пройти через свій власний життєвий цикл.

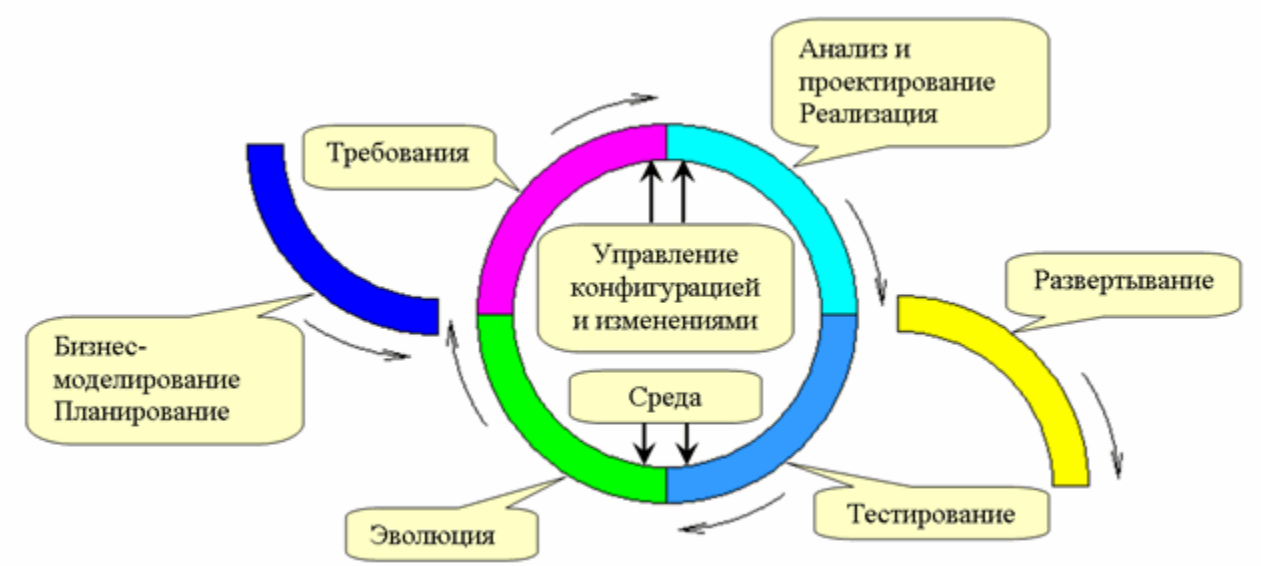


Рисунок 2.1 - Життєвий цикл продукту по RUP

Тестування зазвичай проводиться циклами, кожен з яких має конкретний список задач і цілей. Цикл тестування може збігатися з итерацией або відповідати її певної частини. Як правило, цикл тестування проводиться для конкретної збірки системи.

Життєвий цикл програмного продукту, який зображений на рисунку 2.2 складається з серії щодо коротких ітерацій. Ітерація - це є завершений цикл

розробки, приводить до випуску кінцевого продукту або деякої його скороченою версією, яка розширюється від ітерації до ітерації, щоб, врешті-решт, стати закінченою системою. Кожна ітерація включає, як правило, завдання планування робіт, аналізу, проектування, реалізації, тестування та оцінки досягнутих результатів. Однак співвідношення цих завдань може істотно змінюватися. У відповідність із співвідношенням різних задач в ітерації вони групуються в фази. У першій фазі - Початок - основна увага приділяється завданням аналізу. У ітераціях другої фази - Розробка - основна увага приділяється проектування і випробування ключових проектних рішень. У третій фазі - Побудова - найбільш велика частка задач розробки і тестування. А в останній фазі - Передача - вирішуються в найбільшій мірі завдання тестування і передачі системи замовнику.

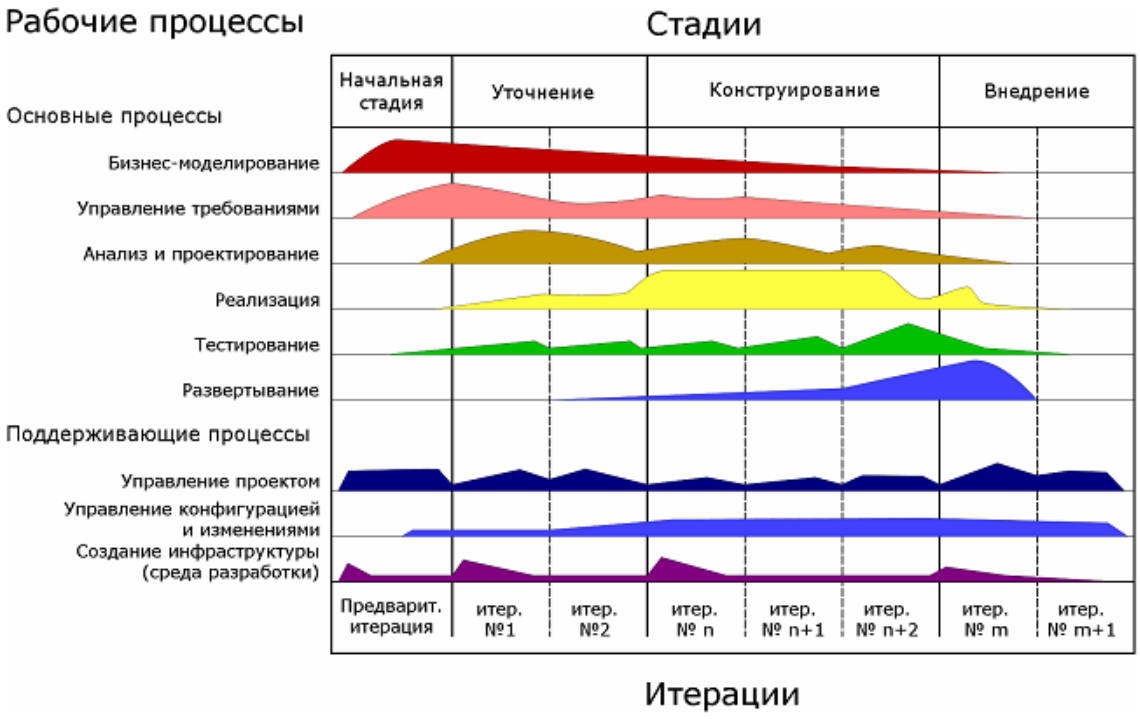


Рисунок 2.2 - Ітерації життєвого циклу програмного продукту

Кожна фаза має свої специфічні цілі в життєвому циклі продукту і вважається виконаним, коли ці цілі досягнуті. Все ітерації, крім, ітерацій фази Початок, завершуються створенням функціонуючої версії розроблюваної системи [6].

2.3 Місце тестування в процесі розробки ПО

Структура фірми - розробника програмного забезпечення відображає етапи життєвого циклу програмного засобу. Те чи інше підрозділ забезпечує виконання робіт по одному або декількох етапах життєвого циклу програмного забезпечення. Аналітичний відділ. До завдань аналітичного відділу входять:

- визначення концепцій і функціонального спрямування розвитку програмного продукту;
- проведення передпроектного обстеження;
- визначення функціональних можливостей системи;
- визначення (спільно з розробниками) технічних вимог до системи;
- опис бізнес-процесів предметної області в термінах, зрозумілих розробникам (Постановки завдань і специфікації на розробку);
- написання постановок задач і специфікацій на доопрацювання програмного засобу при зміні законодавства, вимог клієнтів, розширенні функціональних можливостей продукту;
- контроль процесу реалізації нових можливостей в програмних продуктах компанії.

Відділ документації. Часто даний відділ не виділяється в відокремлену структуру, він може входити, наприклад, до складу аналітичного відділу. До завдань відділу входять написання технічної документації для кінцевого користувача, відстеження змін в програмному засобі і актуалізація в документації.

Відділ розробки. Це ключовий відділ для фірми. Якщо без інших відділів часто можна обійтися, то без відділу розробки не можна. У його завдання входять:

- визначення (спільно з аналітиками) технічних вимог до системи;
- реалізація базових функцій програмного засобу;
- розширення переліку функцій програмного засобу (реалізація доробок);

- виправлення знайдених помилок;
- адаптація програмного продукту для функціонування в інших умовах (перехід на нову СУБД, нову мову програмування та ін.);
- оптимізація програмного продукту (збільшення швидкодії, надійності та ін.).

Відділ технічної підтримки (гаряча лінія). здійснює консультації користувачів з питань, пов'язаних з установкою і експлуатацією програмного засобу по різним каналам зв'язку (телефон, пошта, електронна пошта).

Відділ тестування. До завдання відділу входять:

- комплексний контроль якості;
- підготовка тестової документації (плани тестування та ін.); □ Виявлення та локалізація помилок у функціонуванні програмних продуктів;
- фіксування и відстеження помилок у функціонуванні програмних засобів;
- перевірка відповідності документації програмного продукту стандартам и реально реалізованім функцій;
- участь в розробці та впровадженні системи якості;
- автоматизація тестування;
- цінка продуктівності розроблювальних програмних ЗАСОБІВ на різніх програмно-апаратних платформах и їх специфічніх змінах.

У деяких компаніях на відділ тестування покладаються сборка та випуск програмного забезпечення (в деяких компаніях цим займається відділ розробки) [7].

Всі відділи компанії взаємодіють між собою, дані взаємодії впорядковані між собою и представляються виробничі технологічні процеси. технологічні про- процеси, як правило, регламентовані внутрішніми документами або внутрішньокорпоратівними стандартами;

Типових технологічних ланцюжків всередині компанії - розробника програмного забезпечення безліч. Як приклад можна розглянути схему взаємодії відділу тестування програмного забезпечення з іншими відділами при виявлені помилки під час функціонування програмного забезпечення у користувача.

2.4 Методологія тестування

У термінології професіоналів тестування (програмного та апаратного забезпечення), фрази «тестування білого ящика» и «тестування чорного ящика» відносяться к тому, чи має розробник тестів доступ до вихідного коду тестованого програмного забезпечення, або ж тестування виконується через інтерфейс або прикладний програмний інтерфейс. При тестуванні методом білого ящика (white-box testing, також говорять - прозорого ящика, воно ж структурне тестування), розробник тесту має доступ до вихідного коду програм та може писати код, який пов'язаний з бібліотеками тестованого програмного забезпечення. На рисунку 2.3 можна побачити схематичне зображення даного методу.

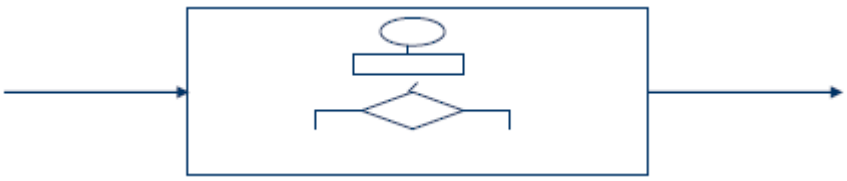


Рисунок 2.3 – Метод білого ящика

Тести створюються на основі знань про структуру самої системи і про те, як вона працює. Критерії повноти засновані на відсотку елементів коду, які відпрацювали в ході виконання тестів. Для оцінки ступеня відповідності вимогам можуть залучатися додаткові знання про зв'язок вимог з певними обмеженнями на значення внутрішніх даних системи (Наприклад, на значення параметрів викликів, результатів і локальних змінних). При тестуванні методом

чорного ящика, схему якого можна побачити на малюнку 2.4, тестувальник має доступ до програмного забезпечення тільки через ті ж інтерфейси, що і замовник або користувач, або через зовнішні інтерфейси, що дозволяють іншого комп'ютера або іншому процесу підключитися до системи для тестування.

Рисунок 2.4 – Метод чорного ящика

Тестування чорного ящика, націлене на перевірку вимог. Тести для нього і критерій повноти тестування будуються на основі вимог і обмежень, чітко зафіксованих в специфікаціях, стандартах, внутрішніх нормативних документах. Часто таке тестування називається тестуванням на відповідність (conformance testing). Окремим випадком його є функціональне тестування - тести для нього, а також використовувані критерії повноти проведеного тестування визначають на основі вимог до функціональності. Крім методів тестування «білий ящик» і «чорний ящик» розрізняють тестування методом «сірого ящика», схема якого зображена на малюнку 2.5

Рисунок 2.5 – Метод сірого ящика

В даному випадку у людини, який розробляє тест кейси, є деяка інформація про внутрішню структуру програми або про деталі реалізації. Даний метод застосовується в Останнім часом частіше попередніх.

2.5 Рівні тестування програмного забезпечення

Існує класифікація видів тестування, яка заснована на тому рівні деталізації робіт проекту, на який вона спрямована. На малюнку 2.6 зображені рівні тестування. ці ж різновиди тестування можна пов'язати з фазою життєвого циклу, на якій вони виконуються.

Модульне тестування (Unit-testing) - рівень тестування, на якому тестується мінімально можливий для тестування компонент, наприклад, окремий клас або функція. На цьому рівні застосовуються методи «білого ящика». У сучасних проектах модульне тестування («юніт-тестінг») здійснюється розробниками [9].

Модульне тестування призначене для перевірки правильності окремих модулів, поза Залежно від їх оточення. При цьому перевіряється, що якщо модуль отримує на вхід дані, задовольняють певним критеріям коректності, то й результати його коректні. для опису критеріїв коректності вхідних і вихідних даних часто використовують програмні контракти - передумови, що описують для кожної операції, на яких вхідних даних вона призначена працювати, постумови, що описують для кожної операції, як повинні співвідноситися вхідні дані з повертаються нею результатами, і інваріанти, що визначають критерії цілісності внутрішніх даних модуля.

Основний недолік модульного тестування полягає в тому, що проводити його можна, тільки коли перевіряється елемент програми вже розроблений. Знизити вплив цього обмеження можна, готуючи тести (а це - найбільш трудомістка частина тестування) на основі вимог заздалегідь, коли вихідного коду ще немає. Модульне тестування є важливою складовою частиною налагоджувального тестування, виконуваного розробниками для налагодження написаного ними коду.

Інтеграційне тестування (Integration testing) - рівень тестування, на якому окремі програмні модулі об'єднуються і тестуються в групі. зазвичай інтеграційне тестування проводиться після модульного тестування (юніт-тести для модулів повинні бути виконані і знайдені помилки виправлені) [9].

Інтеграційне тестування в якості вхідних даних використовує модулі, над якими було проведено модульне тестування, групує їх в більш великі безлічі, виконує тести, певні в плані тестування для цих множин, і представляє їх в якості вихідних даних і вхідних для подальшого системного тестування. При цьому перевіряється, що в ході спільної роботи модулі обмінюються даними і викликами операцій, не порушуючи взаємних обмежень на таку взаємодію, наприклад, передумов викликаються операцій.

Інтеграційне тестування виконується розробниками використовується при налагодженні, але на більш пізньому етапі розробки.

Системне тестування (System testing) - це тестування програмного забезпечення, виконується на повної, інтегрованій системі, з метою перевірки відповідності системи вихідним вимогам. Системне тестування відноситься до методів тестування «чорного ящика », і, тим самим, не вимагає знань про внутрішній устрій системи.

Системне тестування виконується через зовнішні інтерфейси програмного забезпечення і тісно пов'язане з тестуванням призначеного для користувача інтерфейсу (або через призначений для користувача інтерфейс), що проводяться за допомогою імітації дій користувачів над елементами цього інтерфейсу. Окремими випадками цього виду тестування є тестування графічного призначеного для користувача інтерфейсу (Graphical User Interface, GUI) і призначеного для користувача інтерфейсу Web-додатків (WebUI). Системне тестування виконується інженерами по тестуванню.

Приймальне тестування (Acceptance testing) - це тестування готового продукту кінцевими користувачами на реальному оточенні, в якому буде функціонувати тестоване додаток. Приймальні тести розробляються користувачами, зазвичай, у вигляді сценаріїв. Для того, щоб знайти більше помилок рекомендується планувати не тільки системне тестування і приймальне, але і модульне і інтеграційне [5].

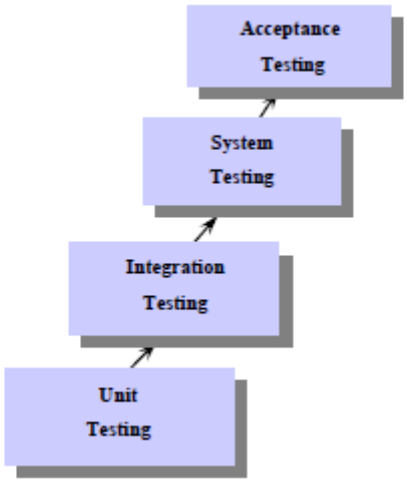


Рисунок 2.6 – Рівні тестування

Статична тестування (Static testing) - тестування, в ході якого тестована програма (код) не виконується (запускається). Аналіз програми відбувається на основі вихідного коду, який вираховується вручну, або аналізується спеціальними інструментами.

Приклади статичного тестування:

- огляди (Reviews);
- інспекції (Inspections);
- наскрізні перегляди (Walkthroughs);
- аудити (Audits).

Динамічне тестування (Dynamic testing) - тестування, в ході якого тестована програма (код) виконується (запускається).

- Альфа-тестування - тестування в процесі розробки.
- Бета-тестування - тестування виконується користувачами (end-users).

Перед тим, як випускається програмне забезпечення, як мінімум, воно повинно проходити стадії альфа (внутрішнє пробне використання) і бета (пробне використання з залученням відібраних зовнішніх користувачів) версій. Звіти про помилки, що надходять від користувачів цих версій продукту, обробляються відповідно до визначених процедур, включають підтверджуючі

тести (будь-якого рівня), що проводяться фахівцями групи розробки. Даний вид тестування не може бути заздалегідь спланований.

Регресійне тестування (Regression testing) - тестування функціональності, яка була вже протестована до внесення будь-яких змін [5]. Після внесення змін до чергову версію програми, регресивні тести підтверджують, що зроблені зміни не вплинули на працездатність решті функціональності додатку. Регресійне тестування може виконуватися як вручну, так і засобами автоматизації тестування.

Визначення успішності регресійних тестів (IEEE 610-90 "Standard Glossary of Software Engineering Terminology ") говорить:" повторне вибіркове тестування системи або компонент для перевірки зроблених модифікацій не повинно призводити до непередбачуваних ефектів ". на практиці це означає, що якщо система успішно проходила тести до внесення модифікацій, вона повинна їх проходити і після внесення таких. Основна проблема регресивного тестування полягає в пошуку компромісу між наявними ресурсами і необхідністю проведення таких тестів в міру внесення кожної зміни. Певною мірою, завдання полягає в тому, щоб визначити критерії "масштабів" змін, з досягненням яких необхідно проводити регресивні тести.

«Смок-тест» (Smoke Testing, «димове тестування») в тестуванні означає мінімальний набір тестів на явні помилки. Димової тест зазвичай виконується самим програмістом; що не проходить цей тест програму не має сенсу віддавати на більш глибоке тестування [5].

2.6 Види тестування

Функціональне тестування (Functional testing) - мета даного тестування полягає в тому, щоб переконатися в належному функціонуванні об'єкта тестування:

- кожне функціональне вимога транслюється в тест-кейси (використовуючи техніки «Чорного ящика») для того, щоб перевірити, що система функціонує в точності, як і описано в специфікації (функціональних вимогах до системи);

- перевірятися, чи всі функціональні вимоги дійсно запрограмовані / реалізовані.

Тестування продуктивності (Perfomance testing) - тестування, яке проводиться з метою визначення, як швидко працює система або її частина під певним навантаженням. Також може служити для перевірки і підтвердження інших атрибутів якості системи, таких як масштабованість, надійність і споживання ресурсів. Існує особливий підвид таких тестів, коли робиться спроба досягнення кількісних меж, обумовлених характеристиками самої системи і її операційного оточення:

- продемонструвати, що система задовольняє критеріям продуктивності при заданих умовах;
- виміряти, яка частина системи є причиною «поганий» продуктивності системи;
- виміряти час реакції на дію користувача, час відгуку на запит, і т.д [11].

Стресове тестування (Stress testing) - зазвичай використовується для розуміння меж пропускної здатності додатки. Цей тип тестування проводиться для визначення надійності системи під час екстремальних або диспропорційних навантажень і відповідає на питання про достатній продуктивності системи в разі, якщо поточна навантаження сильно перевищить очікуваний максимум.

Тестування навантаження (Load testing) - це найпростіша форма тестування продуктивності. Тестування навантаження зазвичай проводиться для того, щоб оцінити поведінку програми під заданої очікуваної навантаженням. Цією навантаженням може бути, наприклад, очікувана кількість одночасно працюючих користувачів додатка, здійснюють вказану кількість транзакцій за інтервал часу. Такий тип тестування зазвичай дозволяє отримати час відгуку всіх найважливіших бізнес-транзакцій. У разі спостереження за базою даних, сервером додатків, мережею і т. д., цей тип тестування може також ідентифікувати деякі вузькі місця докладання [11].

Тестування стабільності (Stability testing) проводиться з метою переконатися в тому, що додаток витримує очікуване навантаження протягом тривалого часу. При проведенні цього виду тестування здійснюється спостереження за споживанням додатком пам'яті, щоб виявити потенційні витоку. Таке тестування виявляє деградацію продуктивності, що виражається в зниженні швидкості обробки інформації і / або збільшенням часу відповіді додатки після тривалої роботи в порівнянні з початком тесту.

Тестування зручності використання (Usability testing) - дослідження, що виконується за метою визначення, чи зручний деякий штучний об'єкт (такий як веб- сторінка, призначений для користувача інтерфейс або пристрій) для його передбачуваного застосування. Таким чином, перевірка ергономічності вимірює ергономічність об'єкта або системи. Перевірка ергономічності зосереджена на певному об'єкті або невеликому наборі об'єктів, в той час як дослідження взаємодія людина-комп'ютер в цілому - формулюють універсальні принципи [11].

Це метод оцінки зручності продукту у використанні, заснований на залученні користувачів в якості тестувальників, випробувачів і підсумовуванні отриманих від них висновків.

При випробуванні багатьох продуктів користувачеві пропонують в «лабораторних» умовах вирішити основні завдання, для виконання яких цей продукт розроблявся, і просять висловлювати під час виконання цих тестів свої зауваження.

Процес тестування фіксується в протоколі (балці) і / або на аудіо-та відеотехніка - з метою подальшого більш детального аналізу.

Якщо юзабіліті-тестування виявляє будь-які труднощі (наприклад складності в розумінні інструкцій, виконання дій або інтерпретації відповідей системи), то розробники повинні доопрацювати продукт і повторити тестування.

Спостереження за тим, як люди взаємодіють з продуктом, нерідко дозволяє знайти для нього більш оптимальні рішення. Якщо при тестуванні використовується модератор, то його завдання - тримати респондента

сфокусованим на завданнях (але при цьому не "допомагати" йому вирішувати ці завдання).

Основну складність після проведення процедури юзабіліті-тестування нерідко представляють великі обсяги і безладність отриманих даних. Тому для подальшого аналізу важливо зафіксувати:

1) мова модератора і респондента;
2) вираз обличчя респондента (знімається на відеокамеру);
3) зображення екрану комп'ютера, з яким працює респондент;
4) різні події, що відбуваються на комп'ютері, пов'язані з діями користувача:
5) переміщення курсору і натиснення на кнопки миші;
6) використання клавіатури;
7) переходи між екранами (браузера або іншої програми).

Всі ці потоки даних повинні бути синхронізовані з тайм-кодами, щоб при аналізі їх можна було б співвідносити між собою.

Поряд з модератором в тестуванні нерідко беруть участь спостерігачі. По мірі виявлення проблем вони роблять свої замітки про хід тестування так, щоб після можна було синхронізувати їх з основним записом. В результаті кожен значущий фрагмент запису тесту виявляється прокоментований в нотатках спостерігача. В ідеалі провідний (тобто модератор) представляє розробника, спостерігачі - замовника (наприклад видавця), а випробувачі - кінцевого користувача (наприклад покупця).

Існує ще один підхід до usability-тестування: для вирішення завдання запропонованої користувачу розробляється "ідеальний" сценарій вирішення цього завдання. Як правило, це сценарій, на який орієнтувався розробник. При виконанні завдання користувачами реєструються їх відхилення від задуманого сценарію для подальшого аналізу. після декількох ітерацій доопрацювання програми і подальшого тестування можна отримати задовільний інтерфейс з точки зору користувача.

Тестування інтерфейсу користувача (UI testing) - тестування графічного інтерфейсу користувача для того, щоб переконатися, що він відповідає прийнятим стандартам і їх вимогам. Перевіряється, як додаток обробляє введення з клавіатури і «мишки» і як відображаються елементи графічного інтерфейсу (текст, кнопки, меню, списки та інші елементи).

Тестування безпеки (security testing) - оцінка уразливості програмного забезпечення до різних атак. Комп'ютерні системи дуже часто є мішенню незаконного проникнення. Під проникненням розуміється широкий діапазон дій: спроби хакерів проникнути в систему з спортивного інтересу, помста розгніваних службовців, злом шахраями для незаконної наживи. Тестування безпеки перевіряє фактичну реакцію захисних механізмів, вбудованих в систему, на проникнення. В ході тестування безпеки випробувач грає роль зломщика. Йому дозволено все:

- спроби дізнатися пароль за допомогою зовнішніх засобів;
- атака системи за допомогою спеціальних утиліт, які аналізують захисту;
- придушення, приголомшення системи (в надії, що вона відмовиться обслуговувати інших клієнтів);
- цілеспрямоване введення помилок в надії проникнути в систему в ході відновлення;
- перегляд несекретних даних в надії знайти ключ для входу в систему.

Тестування локалізації (Localization testing) - багатогранна річ, що має на увазі перевірку безлічі аспектів, пов'язаних з адаптацією продукту для користувачів з інших країн. Наприклад, тестування локалізації для користувачів з Японії може полягати в перевірці того, чи не видасть система помилку, якщо цей користувач на сайті знайомств введе розповідь про себе символами Kanji, а не англійською шрифтом.

Тестування сумісності (Compatibility testing) - вид нефункціонального тестування, основною метою якого є перевірка коректної роботи продукту в певному оточенні. Оточення може включати в себе наступні елементи:

- апаратна платформа;
- мережеві пристрої;
- периферія (принтери, CD / DVD-приводи, веб-камери та ін.);
- операційна система (Unix, Windows, MacOS, ...);
- бази даних (Oracle, MS SQL, MySQL, ...);
- системне програмне забезпечення (веб-сервер, файрвол, антивірус, ...);

браузери (Internet Explorer, Firefox, Opera, Chrome, Safari).

Висновки з розділу

Вище ми розглянули приклади того, навіщо тестування необхідно та які види тестування взагалі бувають. Тестування програмного забезпечення - це:

- процес дослідження ПО з метою отримання інформації про якість продукту;
- процес перевірки відповідності заявлених до продукту вимог і реально реалізованої функціональності, здійснюваний шляхом спостереження за його роботою в штучно створених ситуаціях і на обмеженому наборі тестів, обраних певним чином;
- оцінка системи з тим, щоб знайти відмінності між тим, який система повинна бути і якою вона є.

У широкому сенсі, тестування - це одна з технік контролю якості (Quality Control), яка включає планування, складання тестів, безпосередньо виконання тестування і аналіз отриманих результатів.

Важливо розуміти, що тестування ПО включає не тільки власне проведення тестів, але і багато інших дій, пов'язані з процесом забезпечення якості:

- аналіз і планування;
- розробку тестових сценаріїв;
- оцінку критеріїв закінчення тестування;
- написання звітів;
- рецензування документації (в тому числі і вихідного коду);
- проведення статичного аналізу.

3 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

3.1 Проблеми тестування програмного забезпечення

Ручне і автоматизоване тестування сьогодні грають істотну роль в будь-якої технологічної компанії. Будь то мобільний або веб-додаток або сайт, перевірка коду вкрай важлива. Правильне планування, коли і яке тестування використовувати, допомагає зберігати час і гроші. Обидві методики тестування мають свої переваги і недоліки, їх ми розглянемо нижче.

Ручне тестування може займати багато часу, зате в короткостроковій перспективі заощадить в рази більше грошей. Його вартість залежить тільки від тестувальника, а не інструментів для автоматизації. Ручне тестування можна розглядати як взаємодію професійного тестувальника і софта з метою пошуку багів. Таким чином, під час ручного тестування можна отримувати фідбек, що неможливо у зв'язку з автоматизованою перевірці. Іншими словами, взаємодіючи з додатком безпосередньо, тестувальник може порівнювати очікуваний результат з реальним і залишати рекомендації. Якщо у вас є QA-команда, ручне тестування не буде проблемою.

Плюси ручного тестування:

- Призначений для користувача фідбек. Весь звіт тестувальника може бути розглянутий як зворотний зв'язок від потенційного користувача.

- UI-фідбек. У наш час для користувача інтерфейс грає величезну роль, і тому повністю протестувати його можна тільки вручну. До речі, чи знаєте ви, які 7 елементів інтерфейсу вам краще прибрати з вашого сайту?
- Дешевизна. У короткостроковій перспективі ручне тестування дешевше, ніж інструменти автоматизованої перевірки.
- Тестування в реальному часі. Незначні зміни можуть бути досліджені відразу, без написання коду і його виконання.
- Можливість дослідного тестування. Його метою є перевірка різноманітних можливостей програми. Важливо, що використовуються не заздалегідь складені тест-кейси, а придумані на льоту сценарії.

Мінуси ручного тестування:

- Людський фактор. Хоча UI і може бути протестований тільки вручну, люди часто схильні до неефективності. Деякі помилки можуть залишитися непоміченими.
- Трудомісткість повторного використання. Провести серію стандартних автоматичних тестів простіше, ніж протестувати проект вручну після внесення навіть невеликих змін.
- Неможливість навантажувального тестування. Не можна змоделювати велику кількість користувачів вручну.
- Плюси автоматизованого тестування:
- Можливість навантажувального тестування.
- Можна досить швидко змоделювати велику кількість користувачів.
- Економія часу.
- Ручне тестування великих додатків - довгий і трудомісткий процес, в той час як сценарії пишуться лише один раз.
- Можливість повторного використання.
- Тестовий сценарій, написаний один раз, може бути використаний і в майбутньому при черговому оновленні проекту.

Мінуси автоматизованого тестування:

- UI-тестування. Автоматизоване тестування не може в повній мірі покрити вимоги до призначеного для користувача інтерфейсу.
- Відсутність «людського погляду». Можливо існування помилок, які помітить тільки людина.

Висновок: На жаль, для перевірки функціональності сайту на правильність програми не придумали і найближчим часом думаю, що не придумають. Є різні варіанти для перевірки правильності посилань на сайті, верстки веб сторінок, кросбраузерності, але в них є купа мінусів через які використовувати дану програму не захочеться. Про них буде написано нижче. Універсальної програми для якісного тестування не придумати. Потрібно відштовхуватися від певного об'єкта, над яким працюєш. Під нього писати і підтримувати автоматизовані тести, а після коректо їх запускати.

3.2 Katalon Studio

Katalon Studio - це ефективний інструмент для автоматизації процесу тестування веб-додатків, мобільних додатків і веб-сервісів. Katalon Studio є нащадком таких фреймворків, як Selenium і Appium. Він перейняв у останніх безліч переваг, пов'язаних з інтегрованою автоматизацією тестування ПО.

Для початку роботи з даним інструментом ви можете як володіти початковими знаннями в тестуванні ПО, так і бути справжнім гуру своєї справи. Люди, далекі від програмування, можуть з легкістю запустити свій проект по автоматизації тестування (наприклад, запустивши функцію Object Spy для запису тестових скриптів), а для програмістів і досвідчених тестувальників Katalon Studio виявиться корисним з точки зору економії часу при написанні нових бібліотек і підтримки існуючих скриптів. На рисунку 3.1 можна побачити інтерфейс для запису сценарію для тесту.

Katalon Studio може бути інтегрований в CI / CD, він прекрасно працює в зв'язці з популярними інструментами під час тестування ПО: qTest, JIRA, Jenkins і Git. Для нього передбачена приємна функція - Katalon Analytics, завдяки якій користувачі отримують повне уявлення про процес тестування.

Для цього передбачені спеціальні звіти, які виводяться на екран користувачів у вигляді метрики, діаграм і графіків. Також існує функція для експорту готових тест кейсів у скрипти різних мов. На рисунку 3.2 наводиться приклад експорту в скрипт Java мови розробки.

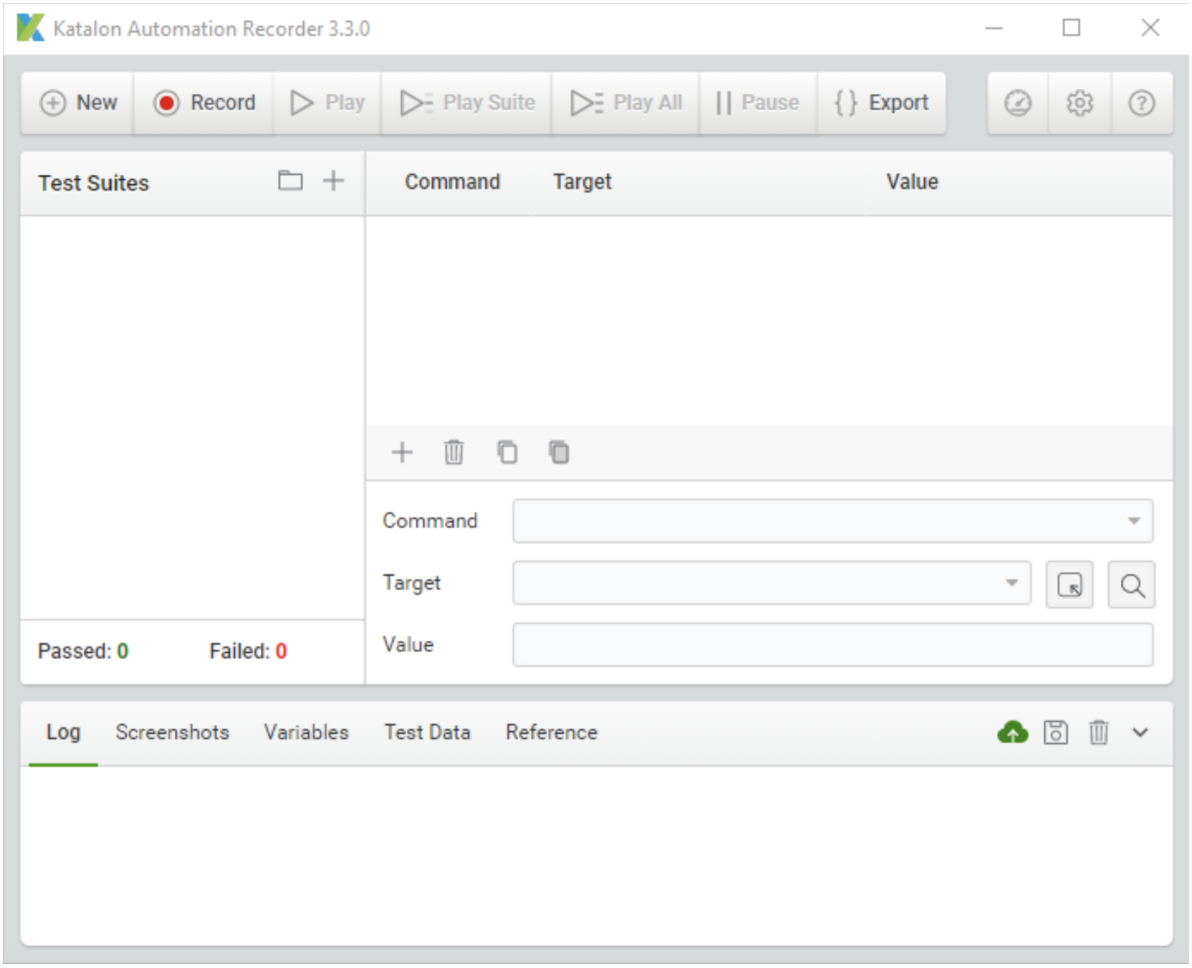


Рисунок 3.1 - Інтерфейс запису сценарію для теста

Недоліки:

- Існує ряд команд, які на даний момент знаходяться на стадії розробки. Наприклад: sendKeys є експериментальною командою. Скоріш за все, в майбутньому ця команда буде замінена на інше - typeKeys. Таким чином, існує вірогідність того, що при використанні експериментальних команд треба буде оновити деякі тестові сценарії.
- В інструменті немає функції відображення базового URL-адреса. В Selenium IDE така функція була дуже зручною для ініціалізації тестових випадків в декількох різних доменах.

- Крім описаних вище недоліків в інструменті також існують і інші помилки. В розділах «Suggestions» (Пропозиції) і «Katalon Automation Records Bugs» (Журнал помилок Automation Katalon) ви можете знайти інформацію про існуючі помилки та способи їх усунення.

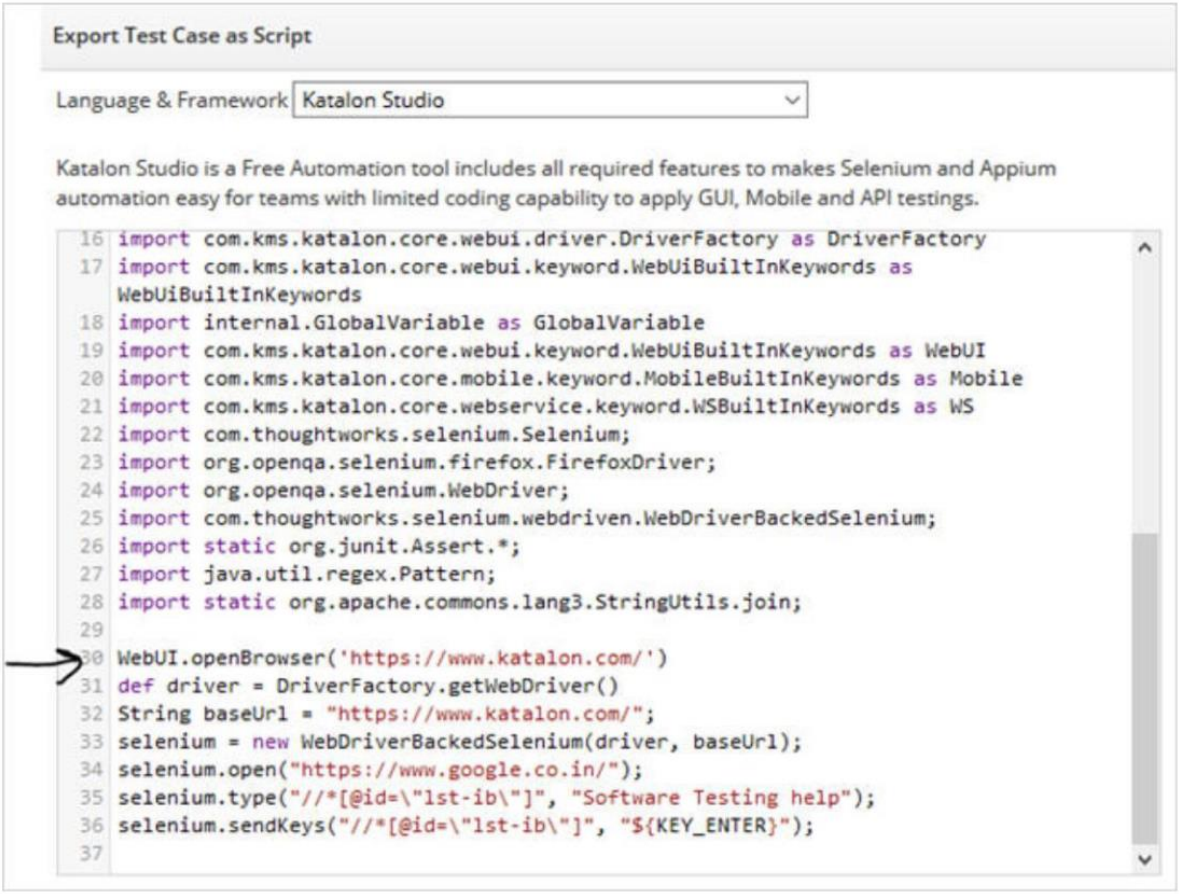


Рис 3.2 – Експорт тестового сценарію в Katalon Studio

3.3 Selenium IDE

Selenium IDE (інтегроване середовище розробки) - це найпростіший інструмент у серії Selenium. Це надбудова Firefox, яка дуже швидко створює тести завдяки функції запису та відтворення. Ця функція аналогічна функції QTP. Це легко встановити і легко навчитися.

Через свою простоту, IDE Selenium слід використовувати лише як інструмент для створення прототипів, а не загальне рішення для розробки та підтримки складних наборів тестів.

Хоча ви зможете використовувати програму Selenium IDE без попередніх знань у програмуванні, ви повинні, принаймні, бути знайомі з HTML, JavaScript та DOM (Document Object Model), щоб використовувати цей інструмент у повному обсязі. Знання JavaScript буде потрібно, коли ми дійдемо до розділу про команду "runScript". На рисунку 3.3 можна побачити приклад того, як структурно виглядає інтерфейс програми та запис тест кейсу.

IDE Selenium підтримує режим автозаповнення під час створення тестів. Ця функція служить двом цілям:

- Це допомагає тестеру вводити команди швидше.
- Це обмежує користувача введення невірних команд

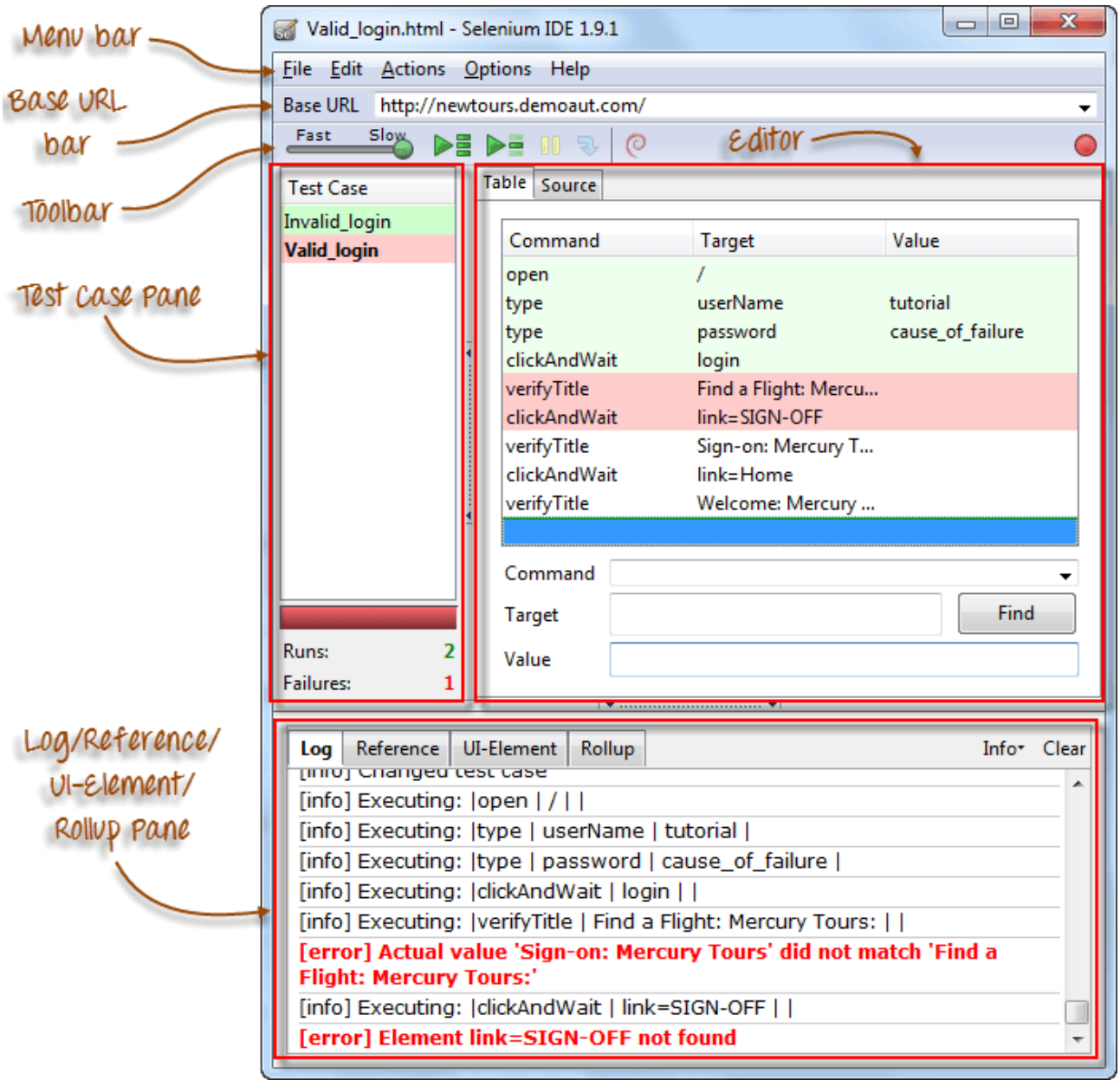


Рис 3.3 – Запис тест кейсу в Selenium IDE

Меню файлу містить параметри створення, відкриття, збереження та закриття тестів. Тести зберігаються у форматі HTML. Найбільш корисним варіантом є "Експорт", який зображений на рисунку 3.4. Він дозволяє перетворити ваші тести Selenium IDE у формат файлів, які можуть працювати на пульті управління Selenium та WebDriver Експортувати тестовий випадок, як ..., буде експортувати лише поточний відкритий тестовий випадок. "Export Test Suite As ..." буде експортувати всі тестові випадки в поточному відкритому тестовому наборі.

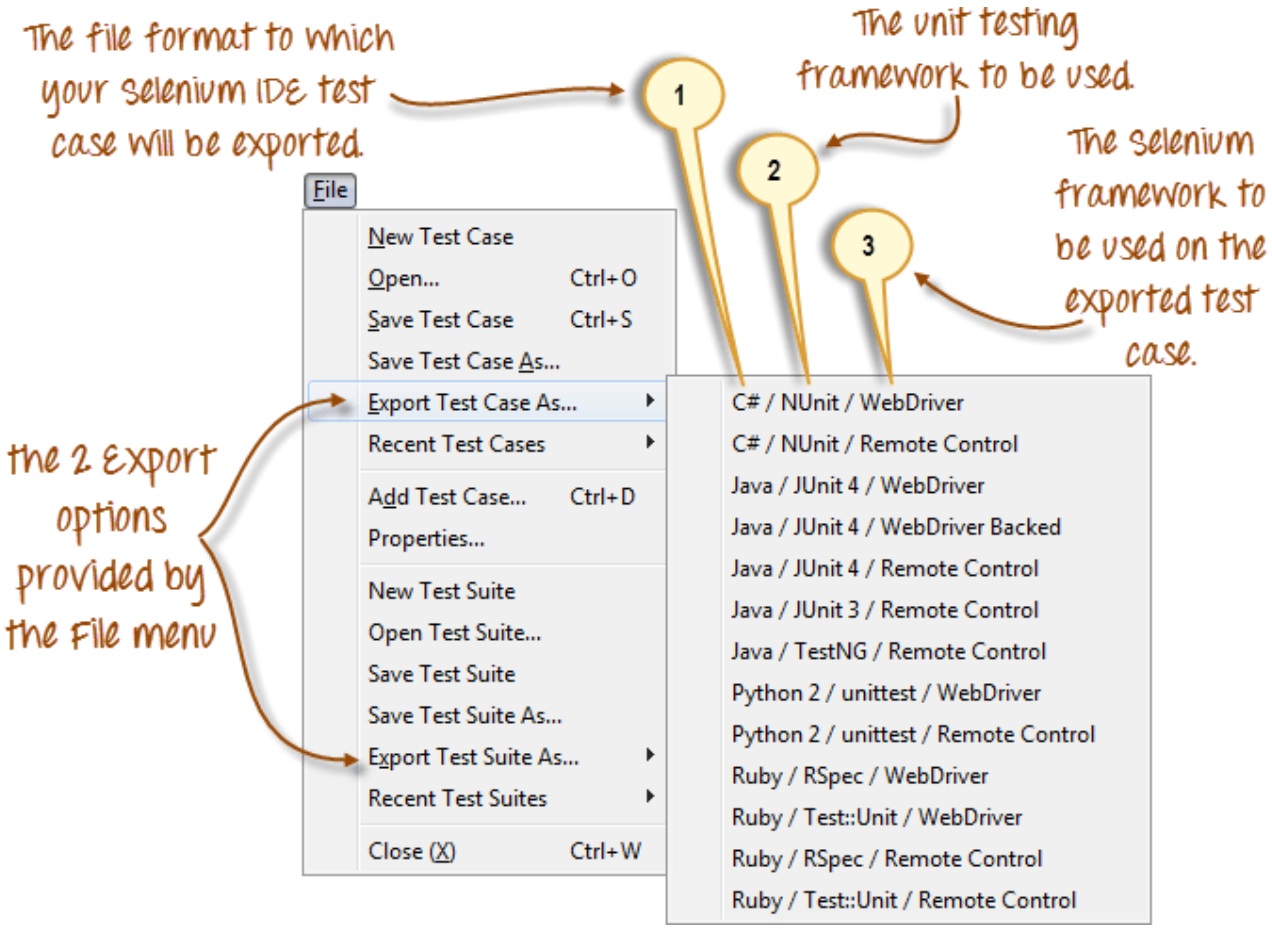


Рис 3.4 – Експорт тестів в Selenium IDE

Висновки з розділу

Перераховані вище програми є хорошим стартом для написання автоматизованих тестів, коли ти ручний тестувальник та коли в тебе немає

досвіду написання на якісь мові програмування. Але з часом ти розумієш, що це дуже кастомні інструменти для автоматизації тестування та якісного покриття функціоналу не буде. Переробляти тестові сценарії у код ніяка машина не зможе так якісно, як людина. Щоб не генерувати код тестів є Selenium Web Driver, про який у дипломній роботі і буде йти мова.

4 ВИБІР ТЕХНОЛОГІЇ ДЛЯ РЕАЛІЗАЦІЇ СИСТЕМИ

4.1 Вибір мови програмування

Так як стоїть завдання про те, щоб запускати тести паралельно, потрібно вибрати багатопоточну мову програмування. В університеті на ранніх курсах ми вчили C #, але після вивчення його мені захотілося спробувати Java.

Обидві мови, C # і Java, - це вдосконалений C ++. Вони були розроблені в різний час, в основному, в конкуруючої середовищі і мають свої подібності та відмінності.

Java був створений під ім'ям Oak Java в 1990 році американською компанією Sun Microsystems. У 1995 році була представлена перша бета-версія цієї мови.

C # з'явився в 2000 році. Незабаром, в 2002, вийшла перша версія фреймворку .Net, який підтримувався цією мовою програмування.

Java був розроблений на основі Objective-C і C, а базою мови C # виступили C ++ і Java. Однак, незважаючи на свою назву, C # ближче до Java, ніж до C ++.

З точки зору розробника, Java і C # мають майже ідентичну структуру. На рисунку 4.1 приклад однакового коду на цих мовах. Обидві мови строго типізовані і об'єктно-орієнтовані. Багато функцій запозичені з синтаксису C ++, проте ці мови простіше для вивчення початківцям.

Java і C # можна порівняти за такими характеристиками:

- синтаксична основа;
- механізм роботи з динамічними даними;
- об'єктні кошти;
- типи даних;
- корисні функції

Code Listing 35: MyShape Package in Java

```
1.  package com.myShape;
2.  public class Line{ }
  *********another source File ****************
1.  package com.myShape;
2.  public class Rectangle{ }
  *********another source File ****************
1.  import com.myShape.Rectangle;
2.   class Test{
3.    public static void main(String arg[]){
4.      Rectangle r = new Rectangle();
5.    }
6.  }
```

Code Listing 36: MyShape Package in C#

```
1.  namespace com.myShape{
2.   public class Line    {   }
3.   public class Rectangle{   }
4.  }
********** another source file **************
1.  using System;
2.  using con.myShape;
3.  class Test{
4.   public static void Main(String []arg){
5.    Rectangle r= new Rectangle();
6.   }
7.  }
```

Рисунок 4.1 – Приклад однакового коду на C# та Java

Обидві мови мають схожу модель роботи з динамічними даними: об'єкти створюються динамічно за допомогою конструкції "new". Середовище виконання відстежує наявність посилань на них. Збирачі сміття періодично очищають пам'ять об'єктів, у яких немає посилань.

У Java і C # є сильні і слабкі посилання на об'єкти. Обидві мови підтримують метод фіналізатор. Через невизначеність моменту видалення об'єкту, фіналізатор не можуть використовуватися для звільнення системних ресурсів, займаних об'єктом. Це змушує створювати додаткові методи для об'єкта "cleaning" і викликати їх явно.

В Java 7 додана структура «try-with-resources», яка забезпечує автоматичне очищення, аналогічну C #.

Java дозволяє реєструвати слухача ("listener"), який буде отримувати повідомлення, коли посилання піддається збірці сміття, що покращує продуктивність класу WeakHashMap.

Код і дані можуть бути описані тільки всередині класу. Інкапсуляція. В Java модифікатор "protected", крім доступу з класів-нащадків, надає їм доступ до всіх класів, що знаходяться в тому ж пакеті, що і власник класу.

У C # для об'єктів, які повинні бути видимими в діапазоні збірки (приблизний аналог пакету Java), був введений окремий модифікатор "internal" (аналог «default» в Java). Модифікатор «protected» зберігає свій первісний зміст від C ++ - доступ тільки від класів-нащадків. Також можливе комбінування модифікаторів "internal" і "protected". Внутрішні класи. Обидві мови дозволяють визначати клас в класі.

В Java внутрішні класи використовуються для емуляції замикань. Вони мають доступ до нестатичних елементів батьківського класу. Також, в межах методу, можна визначити локальні класи, які мають доступ на читання до локальних змінних; безіменні (анонімні) локальні класи, які фактично дозволяють створювати екземпляри об'єктів і інтерфейсів, а також методи перекриття класу безпосередньо в місці їх використання.

У C # є замикання і лямбда. Підхід C # більше нагадує C ++: внутрішні класи доступні тільки для статичних елементів зовнішнього класу. Для доступу в нестатичні елементи необхідно явно вказати екземпляр зовнішнього класу. У C # не підтримуються локальні внутрішні класи. Лямбда-вирази з'являються у версії Java 8.

Методи. В обох мовах методи визначаються функціями класу. Тіло методу локалізовано в описі класу. Підтримуються статичні і абстрактні методи.

У C # існує очевидна реалізація методів інтерфейсу, яка дозволяє будь-якому класу реалізовувати їх окремо від своїх власних методів або давати різні реалізації однойменних методів, що належать двом різним інтерфейсів.

В Java 8 з'являється оператор "default", який дозволяє визначити реалізацію методів інтерфейсу "за замовчуванням". Таким чином, клас інтерфейсу позбавляється від зобов'язання реалізації методів за замовчуванням, але може замістити їх.

Віртуальні методи. C # повторює концепцію віртуальних методів C ++: віртуальні методи можуть бути оголошені з ключовим словом "virtual", інші методи не є віртуальними. Більш того, C # вимагає явного оголошення перекриття віртуального методу в похідному класі за допомогою ключового слова "override».

Типи даних. На рисунку 4.2 порівняльна таблиця Java і C #.

Примітивні типи. Обидві мови підтримують ідею примітивних типів (в C # вони є підмножиною типів-значень) і забезпечують автоматичну "упаковку" ("boxing") і "розпакування" ("unboxing"). Кількість примітивних типів в C # більше, ніж в Java за рахунок беззнакових ("unsigned") цілих типів.
Мова Java, для спрощення, відмовився від більшості беззнакових типів.

У C # можливий розгляд примітивних типів як об'єктів, і це одна з причин, яка робить його популярним.

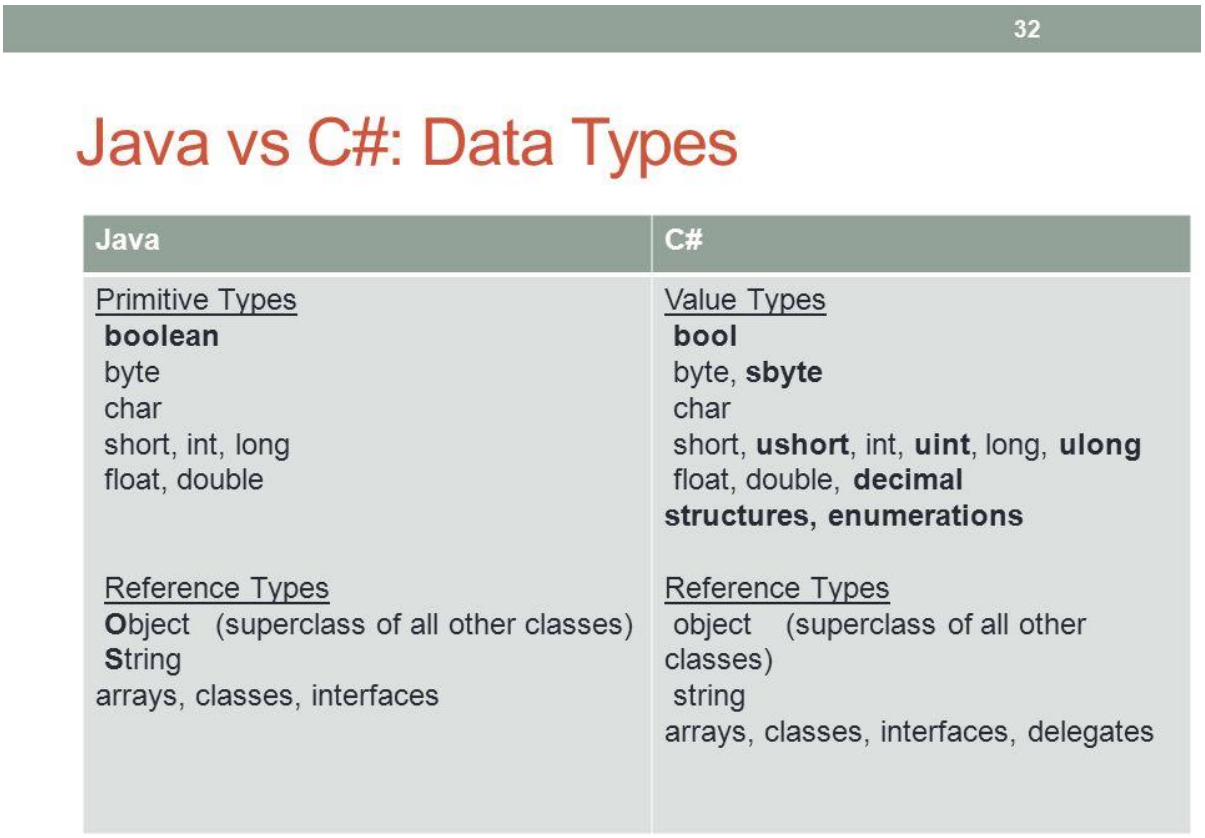


Рисунок 4.2 – Порівняння типів даних Java і C #

Перераховуються типи в C # походять від примітивних цілочисельних типів. В Java перераховуються типи представлені як класи, а їх значення - як об'єкти.

В обох мовах типи можуть бути параметрізірованни, і це підтримує парадигму узагальненого програмування. Узагальнення типів в Java - всього лише лінгвістична структура, які реалізуються тільки на компіляторі.

C # вибрав інший шлях. Підтримка узагальнення була інтегрована в віртуальне середовище виконання, що вперше з'явилася в .NET 2.0. Мова тут - це тільки зовнішній інтерфейс для доступу до цих функцій середовища.

Ми розглянули основні характеристики найбільш популярних мов програмування, і тепер саме час порівняти Java і C # через призму їх корисних функцій, а також архітектурну. До речу, архітектурне порівняння зображено на рисунку 4.3.

В Java навпаки, всі відкриті методи, крім статичних, є віртуальними і неможливо перевизначити метод без участі віртуального механізму. Цей підхід синтаксично простіше, так як він забезпечує виклик методу класу, до якого належить об'єкт.

Переваги C #

- Лямбда-функції

  Також, як і анонімні внутрішні класи в Java, C # забезпечує більш ефективні лямбда-функції. Це зручний метод для визначення об'єкта анонімної функції безпосередньо в точці виклику або передачі функції в якості аргументу. Як правило, лямбда-вирази використовуються для інкапсуляції переданих в алгоритми або асинхронних методів декількох рядків коду.
- Делегати

  Делегати використовуються в якості методів для вказівки на інші методи. В Java теж є така функція, але в C # вона простіше.
- оператор перевантаження

  Мова C # дозволяє виконувати перевантаження операторів для їх використання у власних класах. Це дозволяє досягти природного вигляду визначається користувачем типу даних і використовувати як основний тип даних.

Властивості C # дають кожному класу можливість надати загальний спосіб отримання і цим параметром, якщо приховуючи при цьому код реалізації або

верифікації. Це дозволяє даними бути легкодоступними і продовжувати просувати безпеку і гнучкість методів.

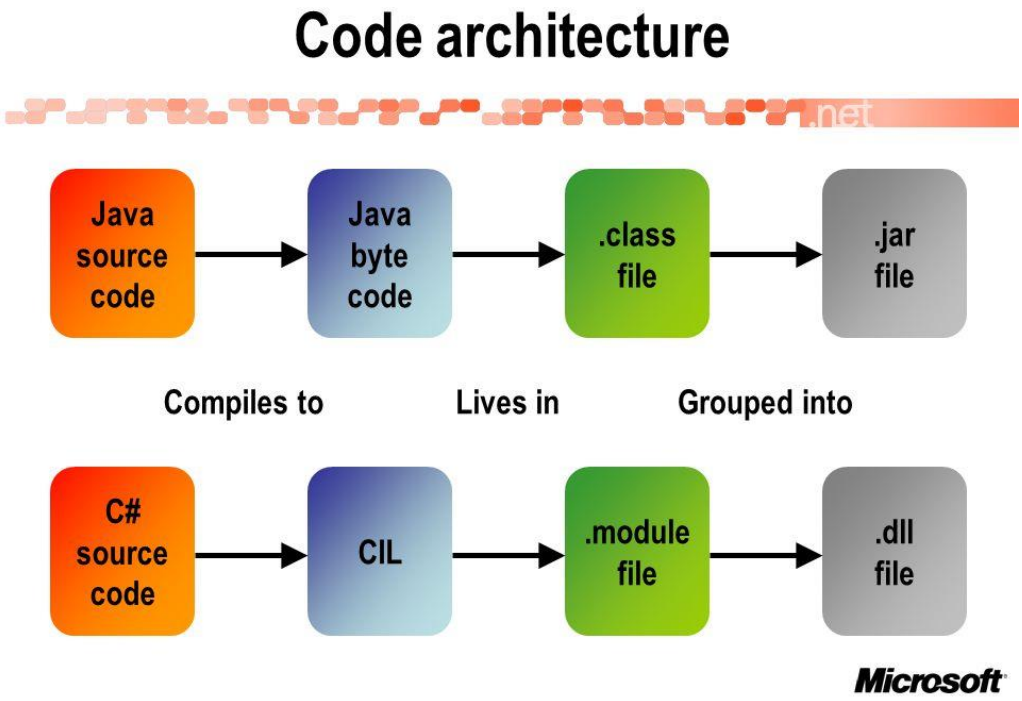


Рисунок 4.3 – Архітектура коду Java vs C#

Переваги Java:

- Сувора статична типізація

  Вона значно зменшує кількість помилок і покращує можливість підтримки коду, особливо при використанні статичного аналізатора.
- Велика історія

  Мова Java існує на ринку з 1995 року, тому сьогодні він широко вивчений і пропонує сотні рішень і технологій.
- Спрощена версія C #

  Всі розробники, які вивчають C ++ і Java, дуже близькі до цієї мови. Java вивчається на основі C ++ і досить легкий. Більш того, Java є простим, передбачуваним і мінімалістичним. Нові функції представлені дуже грунтовно.
- Доступні ціни

Сильна інтегроване середовище розробки, як, наприклад, "IntelliJ IDEA" - вершина еволюції засобів розробки, і вона набагато дешевше, ніж Visual Studio Ultimate для C #.

- Крос-платформенность
  Додатки працюють буквально на всьому: від суперкомп'ютерів до смарт-карт. Особливо добре запускаються на безкоштовному Linux і FreeBSD.
- Велика колекція інструментів для тестування, яка доступна тільки для Java
- Достаток бібліотек
  Існує близько 350 гігабайт бібліотек, доступних в Maven Repository. Можна знайти будь-яку для чого завгодно, і, в більшості своїй, безкоштовно!
- Прихильники Java
  Java - найбільш популярна мова програмування, що є причиною великої кількості Java-програмістів. Для нього також існує офіційна сертифікація SCJP (Sun Certified Java Programmer).
- Повна зворотна сумісність
  Java має повну зворотну сумісність з API і ABI ранніх версій. Жодна мова не зможе похвалитися такою сумісністю, навіть C #. Java ефективно використовується в корпоративній сфері для великих проектів, займаючи дуже сильну позицію.

Ці дві мови дуже схожі між собою, але якщо відкрити інтернет і почитати кількість інструментів для тестування для цих мов, то висновок напрошується сам за себе. Java на даний момент більше готова до написання тестів, існує купа різноманітних інструментів.

4.2 Вибір інтегрованого середовища розробки

В даний час створення програмного продукту немислимо без використання такого інструменту, як інтегроване середовище розробки або скорочено IDE (Integrated development environment). За визначенням середовище включає в себе текстовий редактор, компілятор і / або інтерпретатор, засоби автоматизації збирання і відладчик. Однак, потреби сучасного розробника цим не обмежуються, і щоб їм відповідати, можливості IDE постійно розширюються.

Розглянемо два середовища розробки на мові Java - Eclipse IDE і IDEA - і проведемо короткий порівняльний аналіз.

Eclipse являє собою засновану на Java расширяемую платформу розробки. По суті - це просто середовище розробки і набір сервісів для побудови додатків на основі вбудованих компонентів (плагінів). У складі Eclipse є стандартний набір плагінів, в тому числі добре відомий інструментарій - Java Development Tools (JDT).

Eclipse довгі роки впевнено тримав пальму першості за популярністю серед Java IDE. Це пов'язано з тим, що дана середу повністю безкоштовна, з відкритим вихідним кодом, написаному переважно на Java. Відкритість коду дозволяє користувачам середовища писати свої плагіни і змінювати середу під свої потреби і переваги.

IntelliJ IDEA - інтегроване середовище розробки Java (IDE) для розробки програмного забезпечення. Він розроблений JetBrains (раніше відомий як IntelliJ), і доступний в якості платної розширеної і загальної версії. Обидва вони можуть використовуватися для комерційного розвитку. Community edition призначена для JVM- і Android-розробки.

Безкоштовна версія IntelliJ IDEA підтримує Java, Kotlin, Groovy і Scala; Android; Maven, Gradle і SBT; працює з системами контролю версій Git, SVN, Mercurial і CVS. Однак найчастіше цього виявляється недостатньо для веб-і enterprise-розробки.

Однією з найважливіших функцій сучасних середовищ розробки є рефракторінг. «Рефакторинг - процес зміни внутрішньої структури ПО з метою полегшення розуміння її структури і спрощення подальшої модифікації без

зміни зовнішнього поведінки.» Це визначення взято з книги, що поклала початок сучасному уявленню про цей процес - «Рефакторинг - Поліпшення существуещего коду» Мартіна Фаулера (Refactoring: Improving the Design of Existing Code by Martin Fowler, Addison Wesley 1999).

Інструментарій рефакторінга описується як безліч методів, кожен з яких характеризується ім'ям, областю застосування і механізмом перетворення. На рисунках 4.4 та 4.5 приклад рефакторінгу коду на IntelliJ IDEA та Eclipse. Кожне таке перетворення має невеликий розмір і чітку логіку реалізації, що зводить до мінімуму виникнення помилок внаслідок його виконання.

На даний момент найбільш розвиненою, з точки зору рефакторінга і аналізу коду, є IntelliJ Idea, яка утримує пальму першості на протязі декількох років з її гнучкими функціями code inspection і рефакторінга. Цього варто очікувати від IDE, орієнтованої на розробника, що віддає перевагу кодування.

За нею йде Eclipse, яка має великий потенціал, як платформа для багатомовної розробки, великим дисциплінованим спільнотою розробників. Підтримка рефакторинга організована на базовому рівні і легко може бути реалізована для інших мов, також легко можуть бути додані нові методи рефакторинга. Реалізація для Java забезпечена на прийнятному рівні.

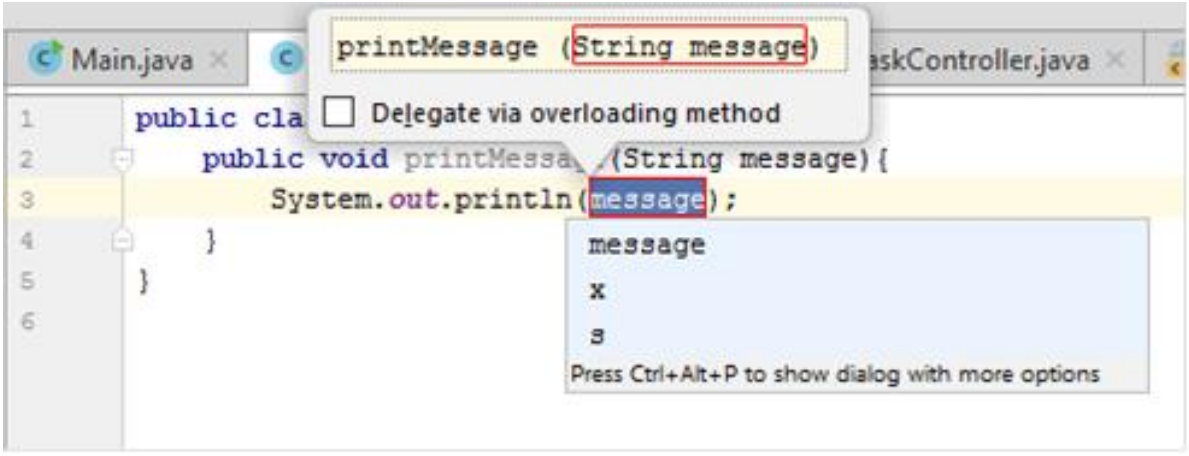


Рис 4.4 – Приклад рефакторінгу в IntelliJ IDEA IDE

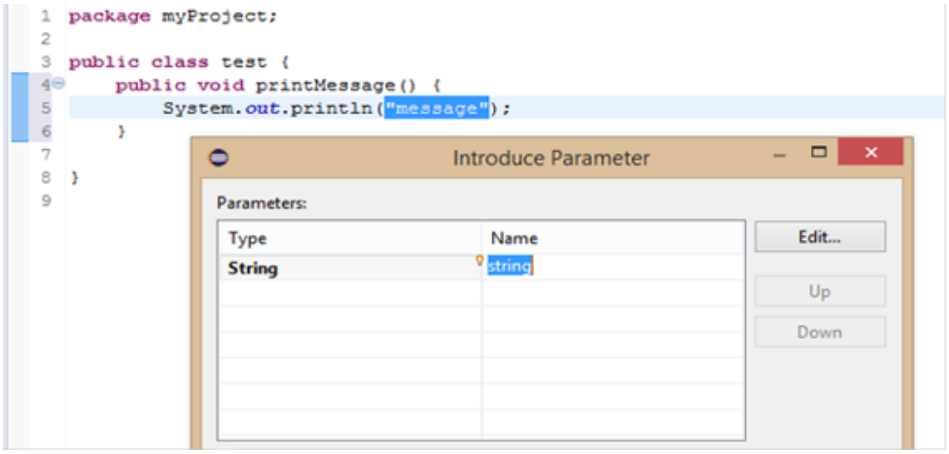


Рис 4.2.5 – Приклад рефакторінгу Eclipse IDE

На відміну від Eclipce IDEA пропонує варіанти для імені параметра. Вона враховує і назва методу, і тип змінної, і значення, і назви подібних змінних в інших місцях, і ті назви, які ви давали подібним змінним раніше.

Особливістю Eclipse і значним плюсом - широка база документації самого різного віку, цінності і корисності. На жаль, виявити невідповідну поточної версії картинку в інструкції, наприклад, із застарілим інтерфейсом і розташуванням кнопок - звичайна справа для цієї IDE. На жаль, проблема запізнілого поновлення документації дуже характерна для будь-яких проектів з вихідним кодом.

Не так давно перевага IDEA над Eclipse було інтегрованим GUI-компоновщиком. Тепер в Eclipse з'явився візуальний редактор (Visual Editor). Компілятор GUI Eclipse - це окремий компонент, який має дуже велику перевагу перед IDEA і не вимагає додаткових метаданих або інших файлів.

Редактор EclipseVisual Editor (VE) 0.5, що підтримує AWT / Swing, доступний для Eclipse 2.1.x. Наступна версія 1.0 для Eclipse 3.0. VE 1.0 можна завантажити як окремий пакет з веб-сайту Eclipse. Ця версія буде підтримувати не тільки AWT / Swing, а й SWT.

Слід сказати, що обидві IDE мають свої переваги і недоліки. Головне достоїнство Eclipse IDE полягає в налагодженні середовища під потреби програміста шляхом впровадження плагінів, а IDEA - в її «інтелектуальності».

Вибір впав на IDEA, це продукт компанії JetBrains. Хто писав колись на C # знає про такий інструмент, як Resharper від JetBrains. Це утиліта, яка допомагає писати код за допомогою різноманітних гарячих клавіш. Так ось в IDEA дана можливість стоїть за умовчанням, це дуже зручно.

4.3 Selenium

Отже, що таке Selenium WebDriver, і чому його немає сенсу порівнювати з TestComplete, QuickTest Pro і іншими інструментами автоматизації тестування. І справа не тільки в тому, що Selenium WebDriver безкоштовний і відкритий - його так само безглуздо порівнювати з іншими безкоштовними інструментами, такими як Sahi або Robot Framework.

Чому? Тому що Selenium WebDriver - це не інструмент для автоматизації тестування. А що ж це таке? На це питання можна дати кілька різних відповідей, спочатку я дам короткі відповіді, а потім - більш докладні. Також спочатку можна подивитись на архітектуру драйверу на рисунку 4.6 для більшого розуміння про що далі буде йти мова.

За призначенням Selenium WebDriver є драйвер браузера, тобто програмну бібліотеку, яка дозволяє розробляти програми, що керують поведінкою браузера.

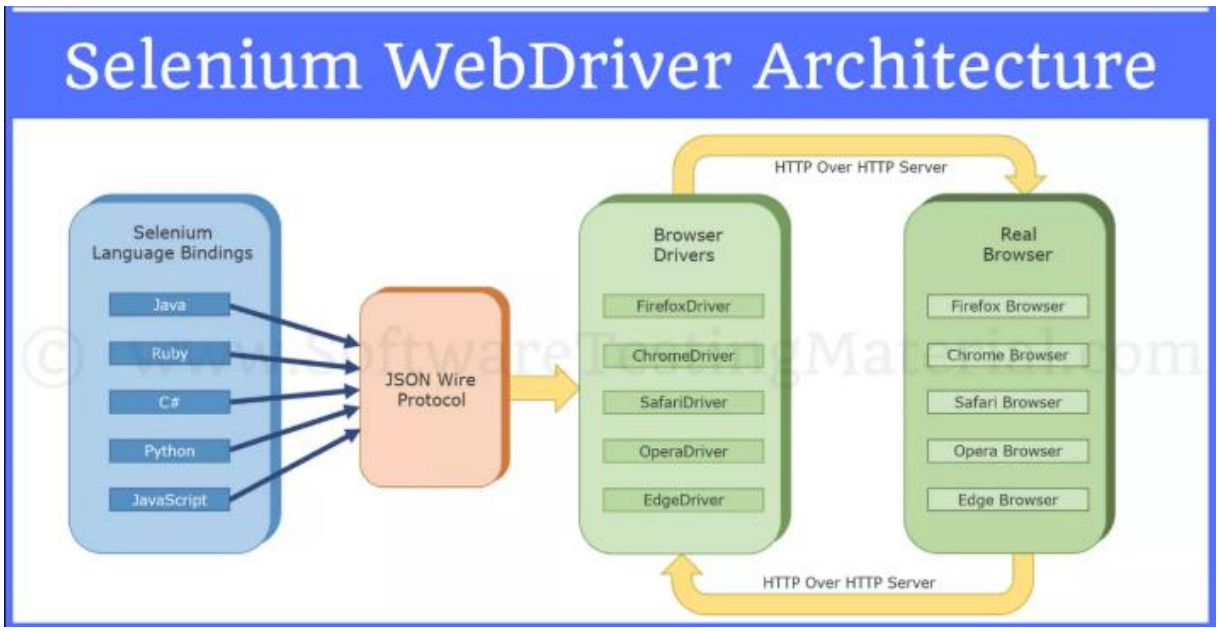


Рис 4.6 – Архітектура Selenium Web Driver

За своєю сутністю Selenium WebDriver є:

- специфікацію програмного інтерфейсу для управління браузером
- реалізації цього інтерфейсу для декількох браузерів
- набір клієнтських бібліотек для цього інтерфейсу на декількох мовах програмування.

Безглуздо порівнювати Selenium WebDriver з «іншими інструментами тестування».

Selenium WebDriver - це драйвер браузера. Напевно кожен, хто стикався з комп'ютерами, навіть не айтішник, знає слово «драйвер». Це така маленька програма, точніше програмна бібліотека, яка дозволяє іншим програмам взаємодіяти з деякими пристроєм. Драйвер принтера дозволяє друкувати що-небудь на принтері. Драйвер диска дозволяє читати і писати дані. Драйвер мережевої карти дозволяє обмінюватися даними з іншими комп'ютерами по мережі.

З драйвером користувачі не працюють безпосередньо. Вони працюють з прикладними програмами, які, за допомогою драйверів, взаємодіють з тими чи іншими пристроями. Драйвер не має призначеного для користувача інтерфейсу. Стривайте, але ж іноді буває призначений для користувача інтерфейс для

налаштування драйвера? Буває. Але це інтерфейс програми для настройки драйвера, а не самого драйвера. Детальна категоризація методів драйверу є на рисунку 4.8 Драйвер має тільки програмний інтерфейс, його призначення полягає в тому, щоб дати можливість прикладним призначеним для користувача програмам взаємодіяти з пристроєм. Деякі методи зображені на рисунку 4.7.

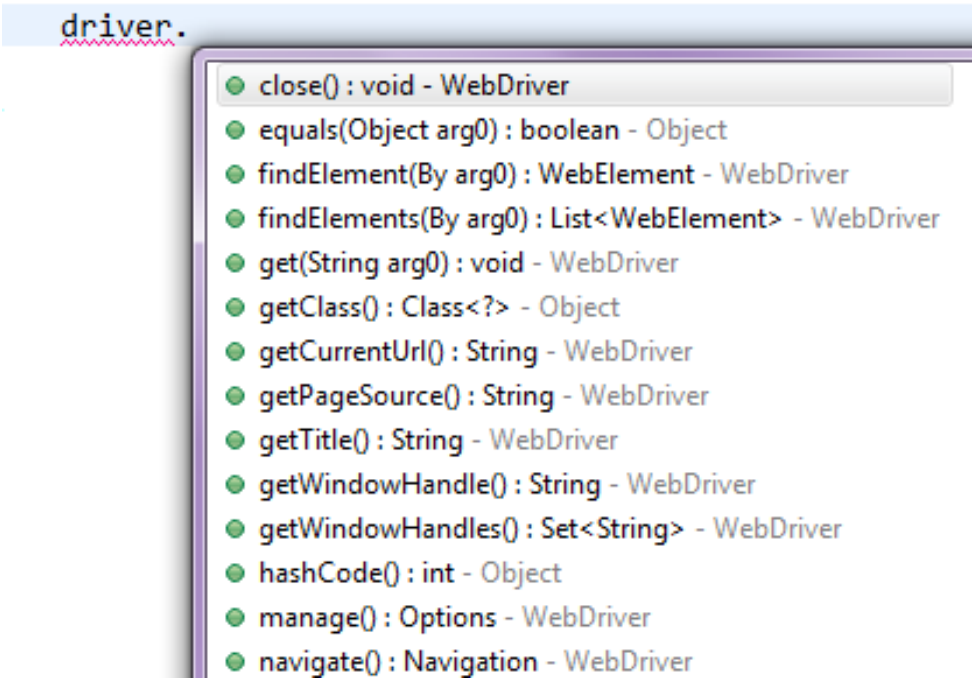


Рис 4.7 – Деякі методи Selenium Web Driver

Selenium WebDriver, або просто WebDriver - це драйвер браузера, тобто не має призначеного для користувача інтерфейсу програмна бібліотека, яка дозволяє різним іншим програмам взаємодіяти з браузером, управляти його поведінкою, отримувати від браузера якісь дані і змушувати браузер виконувати якісь команди.

Виходячи з цього визначення, ясно, що WebDriver не має прямого відношення до тестування. Він всього лише надає Автотест доступ до браузеру. На цьому його функції закінчуються.

Структурування, угруповання і запуск тестів, а також генерацію звітів про тестування, забезпечує фреймворк тестування, такий як JUnit або TestNG для Java, NUnit або Gallio для .Net, RSpec або Cucumber для Ruby і так далі. Розробка тестів ведеться в середовищі Eclipse, Intellij IDEA, Visual Studio, RubyMine і так далі. Збірка здійснюється за допомогою Maven, Gradle, Ant, NAnt, Rake і так далі. Запуск тестів за розкладом і публікацію звітів виконує сервер безперервної інтеграції - Jenkins, CruiseControl, Bamboo, TeamCity і так далі. І все це - самостійні інструменти, що не мають відношення до проекту Selenium.

Втім, в рамках проекту Selenium розробляється не тільки драйвер, але ще кілька супутніх продуктів - Selenium Server дозволяє організувати видалений запуск браузера, за допомогою Selenium Grid можна побудувати кластер з Selenium-серверів. Вони встають в один ряд з перерахованими вище інструментами і фреймворками, тому що також беруть участь в побудові системи запуску тестів. Крім того, є «рекордер», який називається Selenium IDE, він вміє записувати дії користувача і генерувати код, в якому використовується інтерфейс WebDriver для виконання записаних дій.

Але головним у проекті Selenium є саме WebDriver, це ключовий елемент екосистеми Selenium.

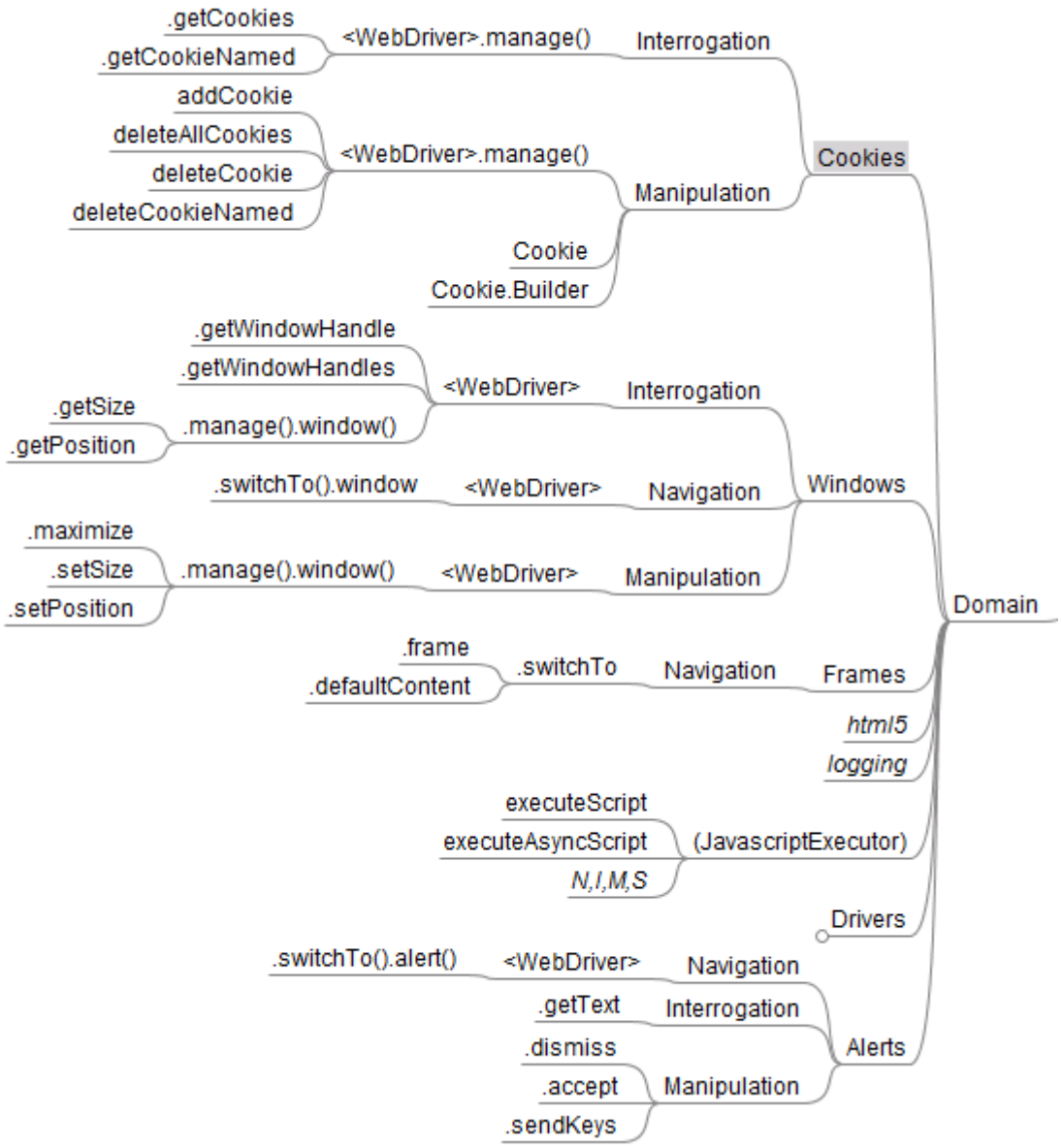


Рис 4.8 – Категоризація методів WebDriver

Усередині кожного комерційного «інтегрованого» інструменту є драйвери браузерів, але вони як правило не можуть бути використані окремо поза цього інструменту. Є і безкоштовні відкриті драйвери - Watir надає доступ до основних браузерам, WatiN має непоганий драйвер для браузера Internet Explorer, Sahi вміє працювати з «великою п'ятіркою» браузерів.

4.4 TestNG

Написані тести потрібно якось запускати. Для запуску існують вбудовані раннери в IDE, але це вкрай незручно. Існує такий фреймворк, як TestNG. Він дозволяє запускати тести з купою різних опцій.

TestNG - це система тестування, натхненна JUnit та NUnit, але введення деяких нових функцій роблять його більш потужним та простішим у використанні. Отже, переваги:

- Виконайте свої тести в довільно великих пулах потоків з різними політиками, доступними (всі методи у власній гілці, одна гілка на тестовий клас тощо ...).
- Перевірте, чи ваш код багатозадачний.
- Гнучка тестова конфігурація.
- Підтримка керування даними (з @DataProvider).
- Підтримка параметрів.
- Підтримуються різноманітні інструменти та плагіни (Eclipse, IDEA, Maven та ін.).
- Вбудовує BeanShell для подальшої гнучкості.
- Типові функції JDK для виконання та реєстрації (без залежності).
- Залежні методи для тестування серверів додатків.

Головна перевага цього фреймворка - це можливість розпаралелювання тестів. Для цього тести повинні бути незалежними.

Розглянемо ієрархію тестів. Всі тести належать до будь-якої послідовності тестів (сюїті), включають в себе деяку кількість класів, кожен з яких може складатися з декількох тестових методів. При цьому класи і тестові методи можуть належати до певної групи. Наочно це виглядає як на рисунку 4.9.

Рис 4.9 - Ієрархія тестів в фреймворку TestNG

У кожного учасника цієї ієрархії можуть бути before і after конфігуратор. Запускається це все в порядку, зображеному на рисунку 4.10.

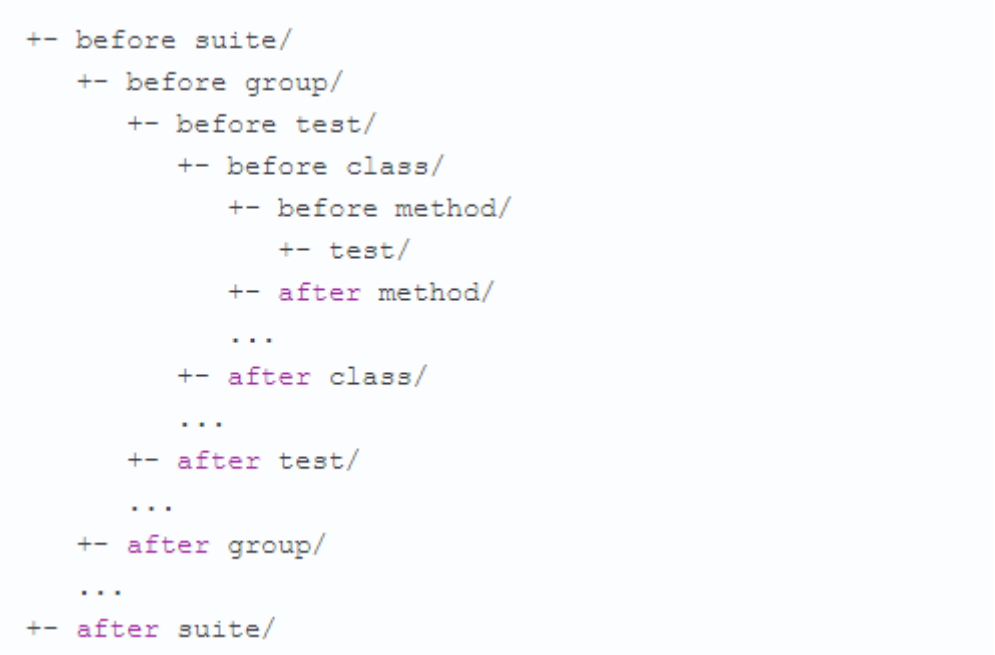


Рис 4.10 – Before/After Configuration

TestNG поширюється як єдиний JAR-файл, який повинен бути доступним у шляху до класу для виконання тестів. Він дуже добре інтегрується з

найпопулярнішими інструментами для створення: Ant, Maven (через Maven Surefire Plugin) і Gradle. TestNG також підтримується всіма основними IDE (в деяких випадках необхідне встановлення додаткового плагіна), може використовуватися з різними мовами JVM (наприклад, Java, Groovy, Scala) і співпрацює з багатьма інструментами якості та тестування (наприклад, інструменти охоплення кодом, насмішкові бібліотеки, бібліотеки збігів). Деякі популярні рішення - наприклад, Spring Framework - забезпечує засоби для полегшення тестування за допомогою TestNG.

4.5 Jenkins

Jenkins - проект для безперервної інтеграції з відкритим вихідним кодом, написаний на Java. Використовується для збірки і постачання програмного забезпечення. У нас є можливість використовувати Jenkins для запуску тестів для кожної нової збірки тестованого продукту.

Одним з основних принципів безперервної інтеграції є те, що побудова повинна бути перевірена. Ви повинні бути в змозі об'єктивно визначити, чи готова певна будівля перейти до наступного етапу процесу зборки, і найбільш зручним способом це зробити - це використовувати автоматичні тести. Без належного автоматичного тестування доведеться тестувати всевручну, що навряд чи в дусі постійної інтеграції.

Для створення тестової збірки нам необхідно встановити додаткові плагіни:

- Maven Integration - для більш зручної інтеграції з Maven
- Allure Jenkins Plugin - для генерації і відображення звіту

Allure Framework - популярний інструмент побудови звітів Автотест, що спрощує їх аналіз. Це гнучкий і легкий інструмент, який дозволяє отримати не тільки коротку інформацію про хід виконання тестів, але і надає всім учасникам виробничого процесу максимум корисної інформації з повсякденного виконання автоматизованих тестів.

Розробникам і тестувальникам використання звітів Allure дозволяє скоротити життєвий цикл дефекту: падіння тестів можуть бути розділені на дефекти продукту і дефекти самого тіста, що скорочує витрати часу на аналіз дефекту і його усунення, що саме і зображено на рисунку 4.11. Також до звіту можуть бути прикріплені логи, позначені тестові кроки, додані вкладення з різноманітним контентом, отримана інформація про таймінгах і часу виконання тестів. Крім того, Allure-звіти підтримують взаємодію з системами безперервної інтеграції і баг-трекінгові системами, що дозволяє завжди тримати під рукою потрібну інформацію про проходження тестів і дефектах.

Рис 4.11 – Приклад успішного білда Jenkins + Allure

Тест-менеджерам Allure дає загальне уявлення про працездатність проекту, дозволяє зрозуміти, які фичи проекту покриті тестами, як згруповані дефекти, який загальний тренд якості проекту.

Висновки з розділу

Отже, для написання коду була вибрана мова Java та Intellij Idea IDE. У написанні тестів на допомогу прийде Selenium Web Driver, котрий вже не

перший рік на ринку і добре себе проявив. Купу необхідних методів для взаємодії з браузером він включив в себе. Паралелізація тестів буде виконуватись з TestNG. Чому саме ця бібліотека, або чого не написати свій кластер для розподілення задач – тому що це просто та перевірено. Запускатись тести будуть з Jenkins. Для кращого відображення результату буде підключено Allure Framework.

5 РЕАЛІЗАЦІЯ СИСТЕМИ

5.1 Опис системи

Часто буває, що після змін в архітектурі, добавлення нових фіч зі сторони розробників або обновлення бібліотек перестає працювати якісь функціонал на сайті.

Кожен раз при оновленні чого-небудь будуть просити ручних тестувальників все перевірити. Але з ростом продукту зростає і кількість завдань. Настає момент, коли тестувальники просто не зможуть встигнути все протестувати.

На допомогу їм приходять автоматизовані тестувальники, які покривають більшу частину функціоналу тестами і частина завдань забирається з плечей тестувальників. Ця система буде відслідковувати, на яких сторінках після внесення змін є помилка.

Метою даної роботи є створення системи для автоматичної перевірки функціоналу тестованого об'єкта та зручним інтерфейсом виводу помилок. Задачі, які треба вирішити для реалізації системи:

- Вибір оптимальної кількості тестів для перевірки продукту Написання тест кейсів
- Вибір мови програмування для написання тестів
- Написання фреймворка для тестування за допомогою Selenium Web Driver
- Написання тестів за допомогою Selenium Web Driver

- Стабільний прогін тестів використовуючи тестовий фреймворк TestNG
- Налаштування Jenkins
- Інтеграція Jenkins з тестами

5.2 Вибір об’єкту для тестування

5.2.1 PDFfiller

Для реалізації ідеї об'єкт для тестування повинен мати деякі якості:

- Доступна деяка функціональність системи
- Зрозумілий графічний інтерфейс
- Стабільна робота сервісу

Вибір впав на PDFfiller. PDFfiller - онлайн сервіс, який допомагає малому, середньому і великому бізнесу економити час і гроші на документообіг. Сотні тисяч користувачів щодня створюють, редагують і автоматично заповнюють корпоративні і особисті документи, підписують контракти і інвойси, а потім експортують їх зручним для них способом. На головній сторінці сайту є слоган цієї компанії, на рисунку 5.1 можна це побачити.

У нього є все ті вимоги, які були описані вище. Щоб достукатися до функціональності сайту, потрібно зареєструватися. Але для підтвердження реєстрації нічого не потрібно, тому відразу ми можемо перейти до роботи з документами. UI частина продукту інтуїтивно зрозуміла, що дає нам з легкістю розуміти, що де і як працює. За час, який я собі виділив на вивчення сайту, ніяких проблем не виявив, тому вважаю, що даний сервіс є стабільним.

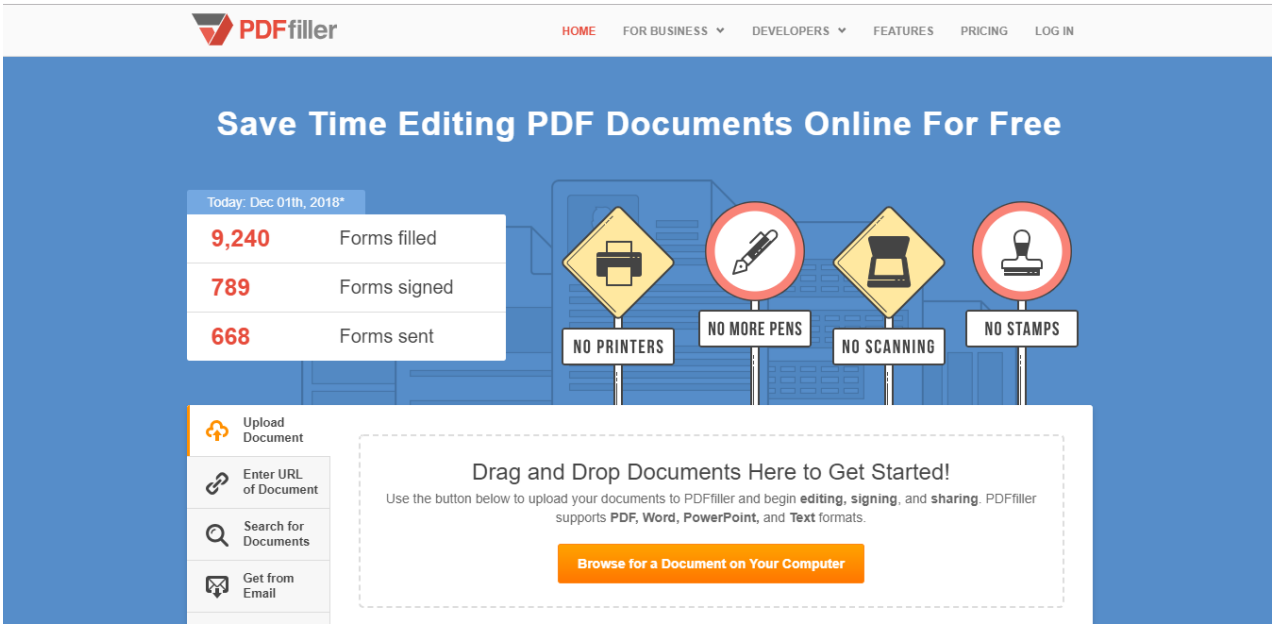


Рисунок 5.1 – Головна сторінка сайту

Кожному зареєстрованому користувачу дається 3 безкоштовних документа для того, щоб протестувати сервіс. З цими документами він може робити майже все те, що є і для платного користувача. На рисунку 5.2 можна побачити цих три документа та дії, які доступні для них.

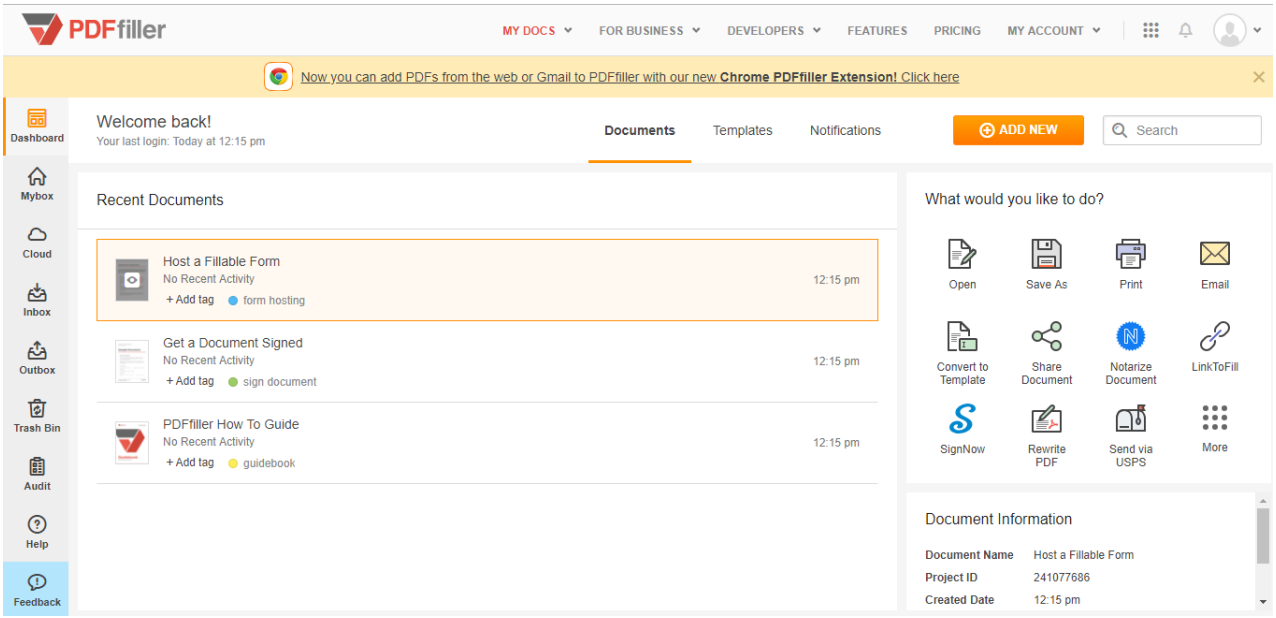


Рис 5.2 – Головна сторінка сайту після авторизації

З лівого боку розташовані різноманітні таби для зручності зберігання документів, а з правого боку знаходяться дії, які можна з документами проводити.

Вивчивши продукт стало зрозуміло, що функціоналу в ньому дуже багато і всі покрити тестами не вийде.

Довгий вибір був щодо функціоналу и вибір впав на експорті документів. Експорт в даному сервісі займає дуже важливу роль. Типовий сценарій користувача системи такий:

1. Знаходить або завантажує потрібну йому форму
2. Змінює її
3. Експортує

Вибір впав на експорт Save As, котрий є найпопулярнішим серед усіх експортів.

5.2.2 Save As

На головній сторінці після реєстрації користувач може попасти на багато варіантів експортів. Всі вони графічно схожі. Перейшовши зі сторінки MyDocs на експорт Save As користувач попадає на сторінку зображену на рисунку 5.3.

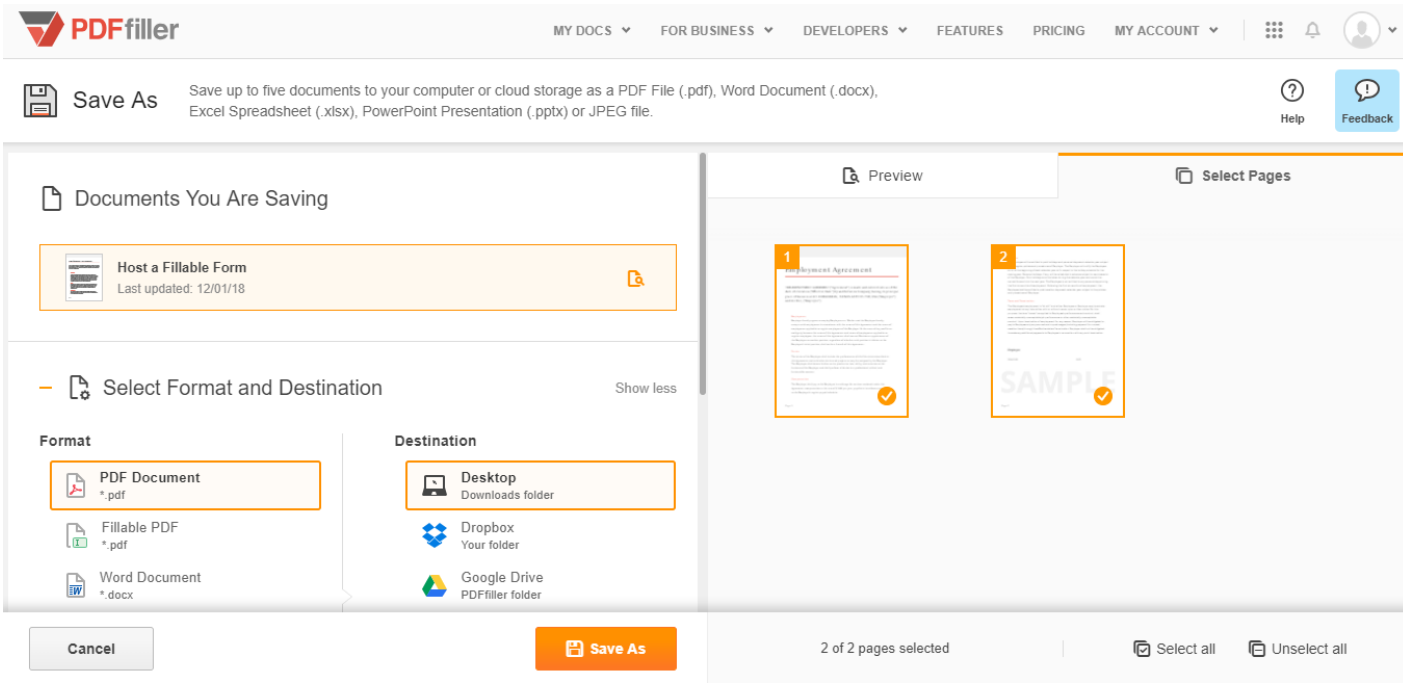


Рисунок 5.3 – Сторінка Save As

На сторінку Save As можна потрапити з декількома документами. Написано, що можна зберегти до 5 документів, але 5 ми не зможемо перевірити, так як у нас немає підписки, а безкоштовних документа всього 3.

Є можливість зберігати в 5 різних форматів: Pdf, Word, Excel, Powerpoint, Image. Зберегти можна на комп'ютер або ж в різні хмарні сховища, такі як: Dropbox, Google Drive, Box, OneDrive, що показано на рисунку 5.4.

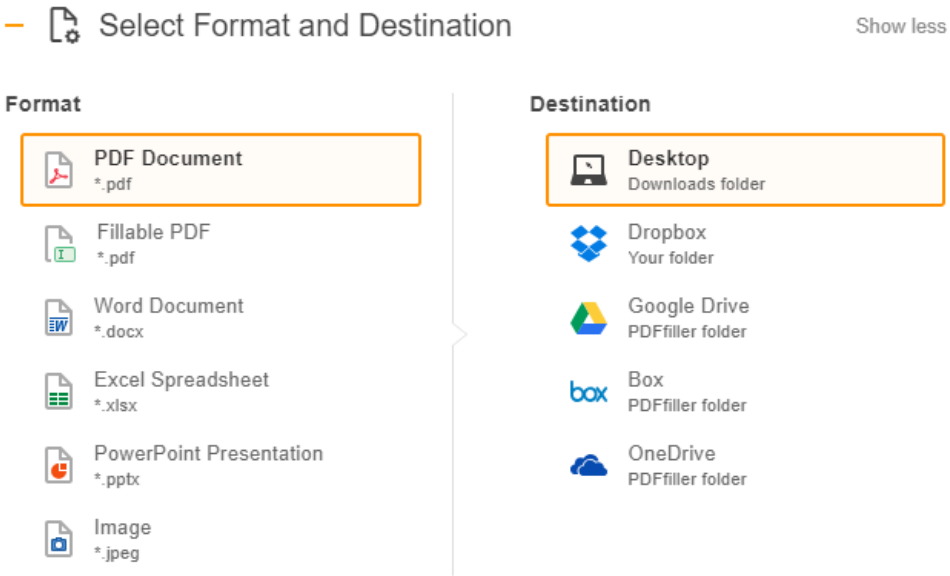


Рисунок 5.4 – Вибір формату та місця збереження

Є можливість збереження частини документа, вибираючи потрібні сторінки в табі Select Pages. Можна вибрати все за допомогою кнопки "Select All", а можна прибрати вибір всіх натиснувши кнопку "Unselect All", рисунок 5.5.

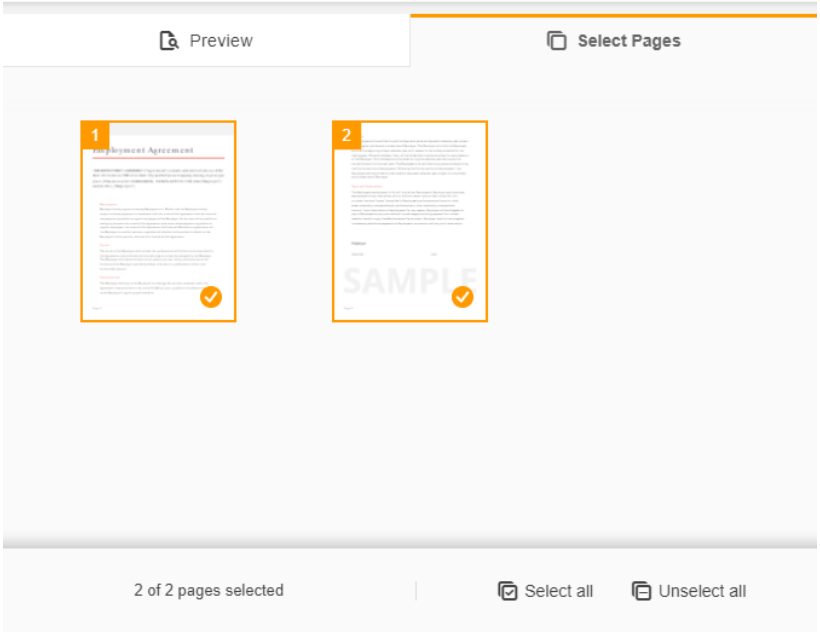


Рис 5.5 – Select Pages Tab

Користувач має можливість подивитися попередньо подивитись свій документ по кожній сторінці за допомогою Preview Tab, рисунок 5.6.

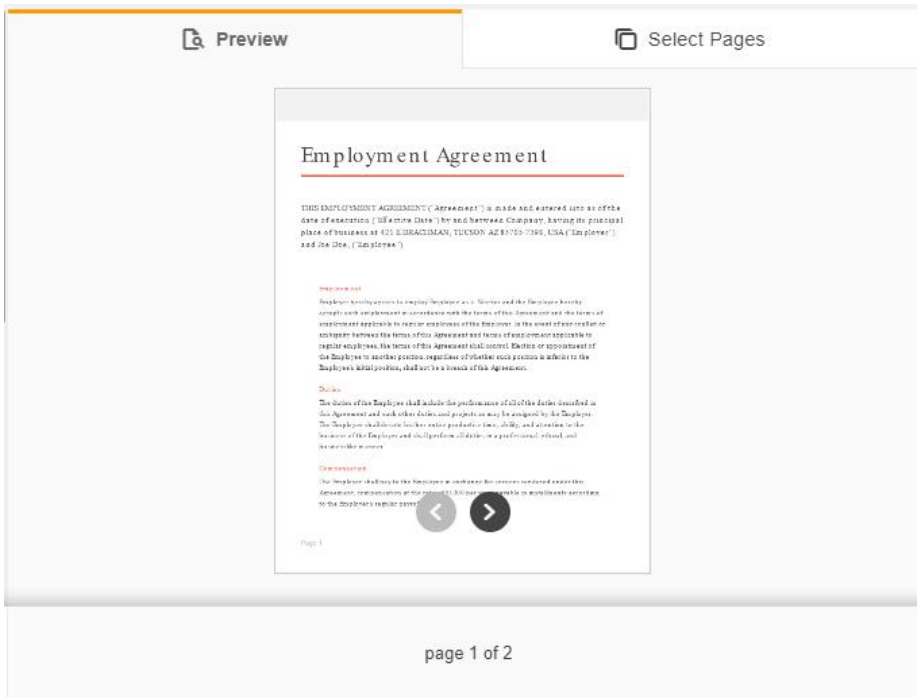


Рис 5.6 – Preview Tab

PDFfiller дає можливість зберігати документи з різними опціями. Дві з них є на рисунку 5.7. З Fillable PDF опцією користувач може зберегти документ з заповнюваними полями. Знаходиться вона у виборі формату. Відкривши документ за допомогою PDF редактора людина зможе відразу ж заповнити його даними і зберегти.

Save Content Only дозволяє зберігати тільки контент, який користувач внесе до документу. Цією функцією дуже часто користуються в Америці, щоб друкувати вже на готових бланках.

Add PDFfiller to File Name додає назву сервісу в назву файлу. Мабуть, це зроблено для того, щоб було видно відміну завантажених файлів в папці.

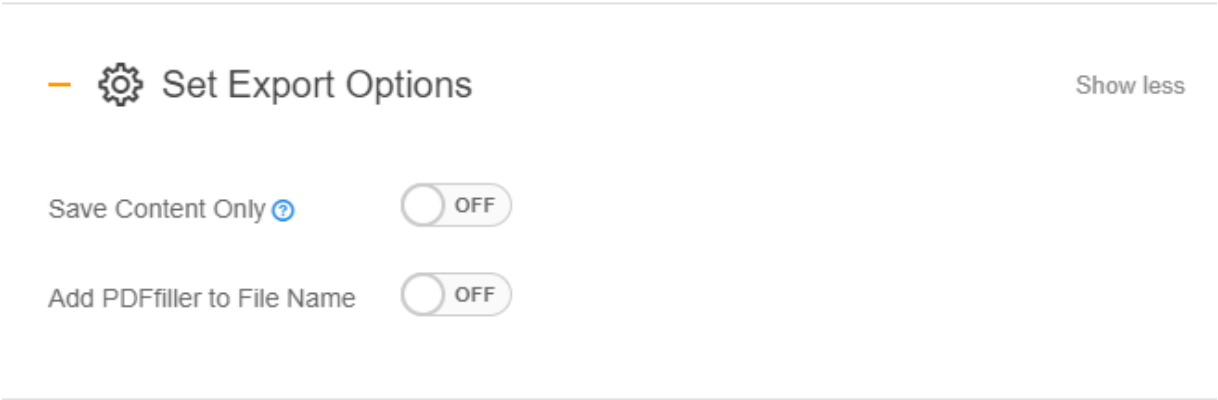


Рис 5.7 – Set Export Options Panel

Існує можливість зберігати файл, поставивши на нього пароль. Цікава функція для безпеки, що зображена на рисунку 5.8 При відкритті такого документа програма запросить пароль.

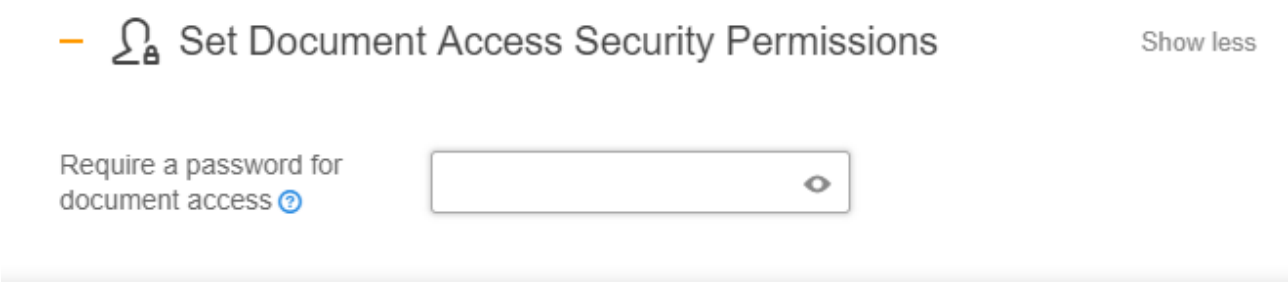


Рис 5.8 – Set Export Options Panel

5.2 Вибір оптимальної кількості тестів для перевірки вибраної частини продукту

Щоб покрити обрану частину сервісу я буду писати функціональні автоматизовані UI тести і автоматизовані тести на верстку, використовуючи бібліотеки, описані в попередньому розділі.

Перш за все, потрібно визначитися що треба автоматизувати. Крім вибраного нами функціоналу, доведеться написати тести і на кроки, за якими ми до експорту прийдемо. Реєстрація та авторизація є саме цими кроками. Далі ми потрапляємо на сторінку My Docs, на яку теж потрібно буде написати мінімально тестів. Після вибору документів ми потрапляємо вже на обраний експорт. Ось структурований список того, що нам потрібно перевірити:

1. Реєстрація
2. Авторизація
3. Перехід на експорт (з різною кількістю документів)
4. Перехід на експорт з платним документом
5. Збереження документа/документів з параметрами за замовчуванням
6. Збереження документа в різні формати
7. Збереження документа з різними опціями
8. Збереження частини документа (використовуючи unselect pages button)
9. Select Pages Tab
10. Preview Tab

5.4 Написання тест кейсів

Таблиця 5.1 - Тестовий сценарій: Перевірка форми реєстрації

1	Перевірка форми реєстрації		
Дії	Очікуваний результат	Фактичний результат	Результат тесту
1) Перейти на сайт pdffiller.com	Перехід на головну	Перехід на головну сторінку	Пройшов

2) Натиснути на кнопку “Log In” у верхній частині екрану 3) Натиснути “Register” 4) Ввести email в поле “Email” 5) Ввести пароль в поле “Password” 6) Натиснути на кнопку “Register and Save”	сторінку сайту з документами	сайту з документами	

Таблиця 5.2 - Тестовий сценарій : Перевірка форми авторизації

2	Перевірка форми авторизації		
Дії	Очікуваний результат	Фактичний результат	Результат тесту
1) Перейти на сайт pdffiller.com 2) Натиснути на кнопку “Log In” у верхній частині екрану 3) Ввести email в поле “Email” 4) Ввести пароль в поле “Password”	Перехід на головну сторінку сайту з документами	Перехід на головну сторінку сайту з документами	Пройшов

5) Натиснути на кнопку “Log In”			

Таблиця 5.3 - Тестовий сценарій: Перехід на експорт з одним документом

3	Перехід на експорт з одним документом		
Дії	Очікуваний результат	Фактичний результат	Результат тесту
1) Авторизуватись на pdffiller 2) Перейти на експорт Save As з одним документом	Перехід на сторінку Save As. Відображається один документ.	Перехід на сторінку Save As. Відображається один документ.	Пройшов

Таблиця 5.4 - Тестовий сценарій: Перехід на експорт з двома документами

4	Перехід на експорт з двома документами		
Дії	Очікуваний результат	Фактичний результат	Результат тесту
1) Авторизуватись на pdffiller 2) Перейти на експорт Save As з двома документом	Перехід на сторінку Save As. Відображається два документа.	Перехід на сторінку Save As. Відображається два документа.	Пройшов

Таблиця 5.5 - Тестовий сценарій: Перехід на експорт з платним документом

5	Перехід на експорт з платним документом		
Дії	Очікуваний результат	Фактичний результат	Результат тесту
1) Авторизуватись на pdffiller 2) Перейти до сторінки з топ 100 документами 3) Обрати документ та відкрити його 4) Натиснути кнопку Done 5) Перейти з документом на експорт Save As	Перехід на сторінку оплати	Перехід на сторінку оплати	Пройшов

Таблиця 5.6 - Тестовий сценарій: Збереження одного документа в різні формати

6	Збереження одного документа в різні формати		
Дії	Очікуваний результат	Фактичний результат	Результат тесту
1) Авторизуватись на pdffiller 2) Перейти на експорт Save As з одним документом 3) Вибрати формат (pdf/word/excel/powerpoint, image) 4) Натиснути кнопку Save As	Документ буде збережений на комп’ютер в .pdf форматі з правильним іменем файла	Документ збережений на комп’ютер в .pdf форматі з правильним іменем файла	Пройшов

Таблиця 5.7 - Тестовий сценарій: Збереження одного документа з опцією Fillable PDF

7	Збереження одного документа з опцією Fillable PDF		
Дії	Очікуваний результат	Фактичний результат	Результат тесту
1) Авторизуватись на pdffiller 2) Перейти на експорт Save As з Host a	Документ буде збережений на комп’ютер в .pdf з одним	Документ був збережений на комп’ютер в .pdf з одним полем для заповнювання	Пройшов

Fillable Form документом 3) Вибрати Fillable PDF опцію 4) Натиснути кнопку Save As	полем для заповнювання		

Таблиця 5.8 - Тестовий сценарій: Збереження одного документа з опцією Add PDFfiller to File Name

8	Збереження одного документа з опцією Add PDFfiller to File Name		
Дії	Очікуваний результат	Фактичний результат	Результат тесту
1) Авторизуватись на pdffiller 2) Перейти на експорт Save As з документом 3) Вибрати Add PDFfiller to File Name опцію 4) Натиснути кнопку Save As	Документ буде збережений на комп’ютер з іменем файла + “-pdffiller”	Документ був збережений на комп’ютер з іменем файла + “-pdffiller”	Пройшов

Таблиця 5.9 - Тестовий сценарій: Збереження одного документа з опцією Save Content Only

9	Збереження одного документа з опцією Save Content Only		
Дії	Очікуваний результат	Фактичний результат	Результат тесту
1) Авторизуватись на pdffiller 2) Відкрити документ та добавити текст Test 3) Перейти на експорт Save As з цим документом 4) Вибрати Save Content Only опцію 5) Натиснути кнопку Save As	Документ буде збережений на комп’ютер. При відкритті буде тільки контент, який був вписаний – “Test”	Документ був збережений на комп’ютер. При відкритті відображається тільки контент, який був вписаний – “Test”	Пройшов

Таблиця 5.10 - Тестовий сценарій: Збереження одного документа не всіма сторінками

10	Збереження одного документа не всіма сторінками		
Дії	Очікуваний результат	Фактичний результат	Результат тесту

1) Авторизуватись на pdffiller 2) Перейти на експорт Save As з одним документом 3) Убрати одну сторінку 4) Натиснути кнопку Save As	Документ буде збережений на комп’ютер без однієї вибраної сторінки.	Документ був збережений на комп’ютер без однієї вибраної сторінки.	Пройшов

Таблиця 5.11 - Тестовий сценарій: Перевірка правильного відображення Preview документа

11	Перевірка правильного відображення Preview документа		
Дії	Очікуваний результат	Фактичний результат	Результат тесту
1) Авторизуватись на pdffiller 2) Перейти на експорт Save As з одним документом 3) Відкрити Preview Tab	Відображається перша сторінка документа у Preview Tab	Відображається перша сторінка документа у Preview Tab	Пройшов

Таблиця 5.12 - Тестовий сценарій: Перевірка, що документ не буде збережений, коли жодної сторінки не вибрано

12	Перевірка, що документ не буде збережений, коли жодної сторінки не вибрано		
Дії	Очікуваний результат	Фактичний результат	Результат тесту
1) Авторизуватись на pdffiller 2) Перейти на експорт Save As з одним документом 3) Натиснути на кнопку Unselect All 4) Натиснути на кнопку Save As	Відображається “Oops” попап за текстом “You haven't selected any pages. Select at least one page of the document to proceed.”	Відображається “Oops” попап за текстом “You haven't selected any pages. Select at least one page of the document to proceed.”	Пройшов

Таблиця 5.13 - Тестовий сценарій: Збереження двох документів

13	Перехід на експорт з двома документами		
Дії	Очікуваний результат	Фактичний результат	Результат тесту
1) Авторизуватись на pdffiller 2) Перейти на експорт Save As з двома документом	ZIP файл буде збережений на комп’ютер. У цьому файлі два документа	ZIP файл був збережений на комп’ютер. У цьому файлі два документа	Пройшов

3) Натиснути Save As 4) Натиснути “Download ZIP”			

Таблиця 5.14 - Тестовий сценарій: Збереження одного документа, коли зайшов з двома

14	Збереження одного документа, коли зайшов з двома		
Дії	Очікуваний результат	Фактичний результат	Результат тесту
1) Авторизуватись на pdffiller 2) Перейти на експорт Save As з двома документом 3) Натиснути Save As 4) Натиснути “Download” біля першого документа	Буде збережений на комп’ютер перший документ	Був збережений на комп’ютер перший документ	Пройшов

5.5 Написання фреймворка для тестування

5.5.1 Для чого він потрібен

У цій роботі буде створений свій фреймворк для тестування. Спочатку звучить страшно і складно, але згодом прийде поняття, що це таке. Фреймворк для тестування - це набір готових методів, даних, бібліотек для взаємодії з тестованим об'єктом. Щоб тести були зручні в експлуатації, вони повинні мати логічну і зрозумілу архітектуру.

Архітектура - це деякий набір правил. Наприклад, таких:

- відокремлюємо логіку тестів (опис того, як система повинна себе вести) від тестових даних (їх можна зберігати, наприклад, в файлах або генерувати на льоту випадкові дані, важливо те, що дані окремо, а логіка їх використання окремо

- відокремлюємо логіку тестів (опис того, як система повинна себе вести) від реалізації дій (які кнопочку треба натискати в інтерфейсі програми або які запити відправляти по мережі)

- дії такого-то типу виконуємо через один інтерфейс (скажімо, UI), а дії іншого виду через інший інтерфейс (скажімо, API).

Пишеться він для того, щоб прискорити процес написання тестів. Ось простий приклад. Нам потрібно натиснути на кнопку для відкриття логін форми. Робиться це в одну строчку:
driver.findElement(By.Id("login-form")).click(); Де login-form це локатор потрібної нам кнопки нам кнопки.

Але перед тим як клікнути, потрібно переконатися, що елемент готовий до роботи. А в разі помилки - обробити помилку. Тому коректний код звичайного кліка для відкриття логін форми буде вже виглядати як на рисунку 5.9.

```
Wait wait = new FluentWait(driver)
        .withTimeout(Duration.ofSeconds(15))
        .pollingEvery(Duration.ofMillis(500))
        .ignoring(ElementNotVisibleException.class).ignoring(NoSuchElementException.class).ignoring(StaleElementReferenceException.class);
try {
    wait.until(ExpectedConditions.elementToBeClickable(By.id("login-form")));
    driver.findElement(By.id("login-form")).click();
} catch (TimeoutException e) {
    Logger.error( obj: "Can not click on element");
}
```

Рисунок 5.9 – Метод кліку на елемент

В кожному тесті виконується безліч кліків і якщо цей код буде повторюватися, то це не відповідає нормам ООП. Тому створюється PageBase клас, в який додають всі методи, що повторюються майже кожен тест. У підсумку цей метод зроблений універсальним для всіх і наведений у додатку А, Page Base class Тепер там є параметр "locator" і можна клікати на будь-які елементи.

Рисунок 5.10 – Метод для кліку на елемент з тестового фреймворку

5.5.2. Page Object Pattern

Коли ви пишете функціональні тести з використанням селену, основна частина вашого коду складається з взаємодій із веб-інтерфейсом, який ви тестуєте через API WebDriver. Після вилучення елементів ви перевіряєте певний стан елемента за допомогою різних тверджень і переходите до вибору наступного елемента. Розглянемо приклад з рисунку 5.11.

```java
List<WebElement> zipCodes = driver.findElements(By.id("zipC
for (WebElement zipCode : zipCodes) {
    if (zipCode.getText().equals("12345")){
        zipCode.click();
        break;
    }
}
WebElement city = driver.findElement(By.id("city"));
assertEquals("MyCityName", city.getText());
```

Рисунок 5.11 – Приклад простого тесту

Це простий приклад тесту, який витягує та повторює список поштових кодів, які шукають 12345, натискаючи його та витягуючи міський елемент, очікуючи, що назва міста буде MyCityName.

Навіть за допомогою простого тесту, як це, читаемость дуже погана. Є великий код WebDriver, який затушує мету тесту, що робить його повільним та складним для перетравлення.

З будь-яким інтерфейсом, і, я вважаю, веб-інтерфейси, зокрема, звичайно, що і незначні, і основні зміни в інтерфейсі виконуються часто. Це може бути новий дизайн, реструктуризація полів і кнопок, і це, ймовірно, вплине на ваш тест. Таким чином, ваш тест не вдається, і вам потрібно оновити селектори.

Тепер, якщо у вас є лише один функціональний тест для сторінки з виключенням "щасливого шляху", ви можете не вважати це великою угодою. Однак, якщо у вас є повний комплект регресійних тестів, це абсолютно велика угода.

Тому деякі з типових проблем для такого типу селенових тестів:

- Тестові випадки важко прочитати
- Зміни в користувальницькому інтерфейсі часто порушують декілька тестів у кількох місцях
- Копіювання селекторів як всередині, так і між тестами - не повторне використання

Таким чином, замість того, щоб кожний елемент тестування завантажувався безпосередньо та був нестійким для змін інтерфейсу користувача, шаблон об'єкта сторінки вводить те, що в основному є розв'язувальним шаром.

Ви створюєте об'єкт, який представляє користувальницький інтерфейс, який ви бажаєте протестувати, що може бути цілою сторінкою або значною його частиною. Відповідальність за цей об'єкт полягає в тому, щоб загорнути HTML-елементи та інкапсулювати взаємодії з користувальницьким інтерфейсом, що означає, що саме там будуть йти всі виклики WebDriver.

Описана схема представлена на рисунку 5.12. Тут є більшість веб-елементів. І це єдине місце, яке потрібно змінити, коли змінюється інтерфейс користувача.

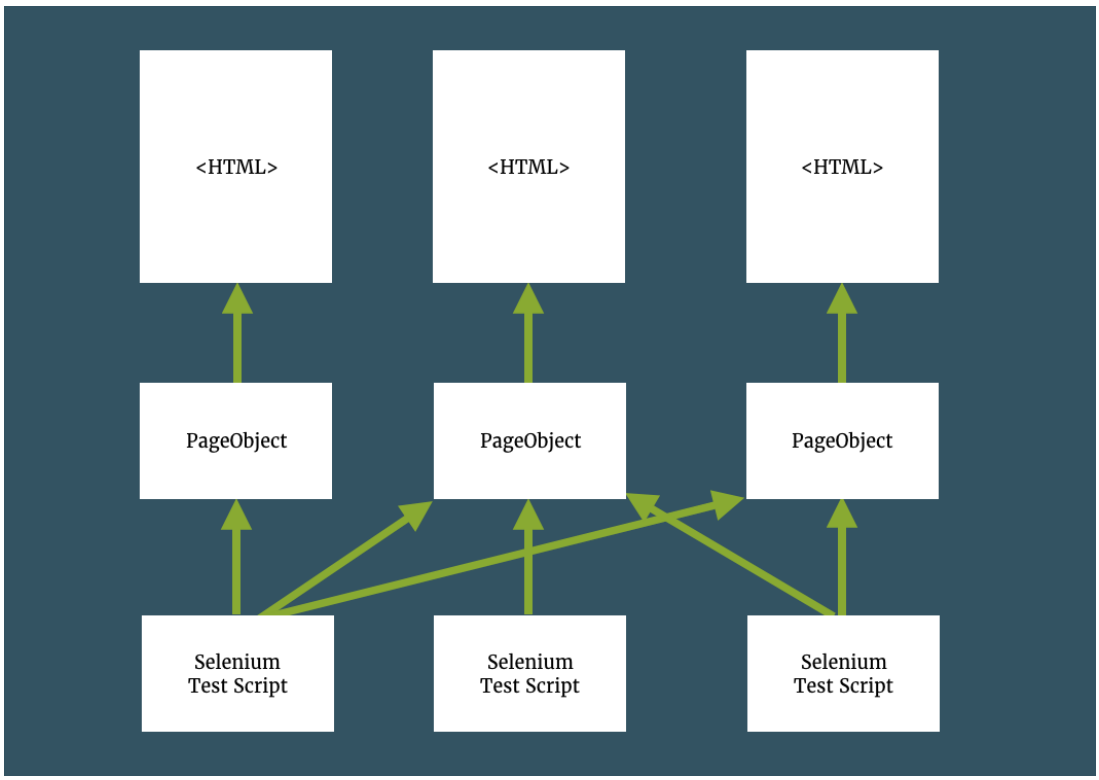


Рисунок 5.12 – Схема роботи при Page Object Pattern

Основні переваги:

- Поділ коду тестів і опису сторінок
- Об'єднання всіх дій по роботі з веб-сторінкою в одному місці

Для прикладу опишем сторінку експорту Sms. Створимо клас з описом цієї сторінки, використовуючи шаблон Page Object. На рисунку 5.13 можна побачити, що у цей клас винесено локатори для взаємодії, основні методи. В тесті потрібно буде тільки визвати об’єкт сторінки і працювати з її методами.

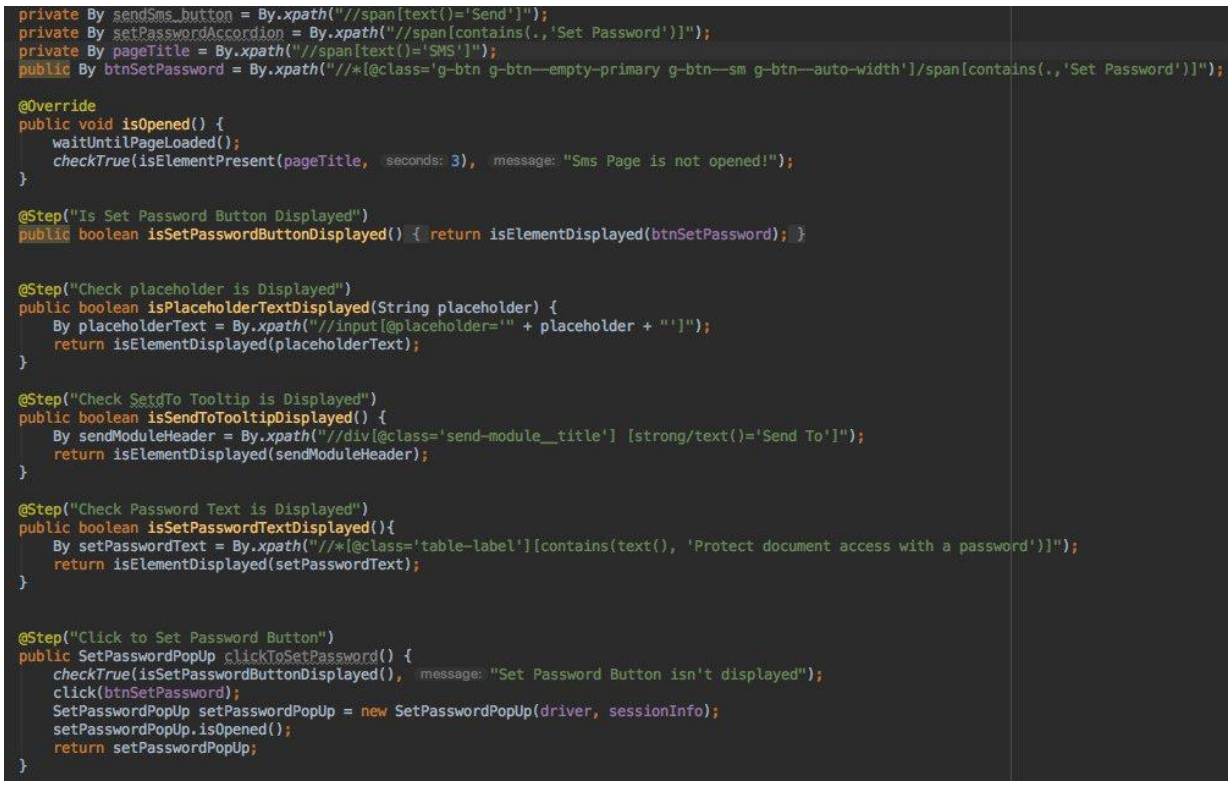


Рисунок 5.13 – Приклад описаної сторінки

5.5.3. Архітектура фреймворку

Є два основні класи, які беруть участь при розробці тестів. TestBase і BasePage. В абстрактному класі BasePage, як я і говорив вище, зібрані всі методи для роботи зі сторінкою. Кожна описана сторінка успадковується від цього класу. Це в рази прискорює процес створення якісного класу сторінки. У додатку Б можна знайти схему «Структура проекту у інтегрованій середі розробки», а також діаграму класів розробленого фреймворку.

У TestBase класі зібрані методи для підняття тесту. Саме в цьому класі викликається драйвер браузера. Тести будуть запускатись тільки в Chrome браузері, але я додав можливість запустити тести на Internet Explorer і Mozilla Firefox. Кожен написаний тест успадковується від цього класу.

Також у фреймворку є деяка особливість пов'язана з перевірками в тестах. Часто буває, що тест не може пройти до кінцевої перевірки і падає раніше, ніж потрібно. Це пов'язано з багатьма факторами, одним з яких є довгий відповідь сервера якогось проміжного кроку. Для таких випадків було придумано Check клас з методами і стандартний assertTrue метод, код яких наведений на

рисунках 5.14 та 5.15 Алгоритм роботи виявлення помилок при тесті наведений у додатку Б.

checkTrue / checkFalse / checkEquals використовується в класах сторінок і якщо перевірка не проходить, то тест валиться помилково AssertionException (Тест стає червоним, то є якийсь проміжний функціонал зламався). А основну перевірку роблю за допомогою методу assertTrue, який повертає fail () і тест стає жовтим (тобто не пройшов головну перевірку). Таким чином стає зрозумілим, зламаний саме тест або якісь кроки.

```
public static void checkTrue(boolean condition, String message) {
    if (!condition) {
        String errorMessage = "Assert True: " + message + ", expected [true] but found [false]";
        throw new AssertionException(errorMessage);
    }
}

public static void checkFalse(boolean condition, String message) {
    if (condition) {
        String errorMessage = "Assert False: " + message + ", expected [false] but found [true]";
        throw new AssertionException(errorMessage);
    }
}

public static void checkEquals(boolean actual, boolean expected, String message) {
    if (actual != expected) {
        String errorMessage = "Assert Equals: " + message + ", expected [" + expected + "] but found [" + actual + "]";
        throw new AssertionException(errorMessage);
    }
}
```

Рисунок 5.14 – Check методи класа Check

```
static public void assertTrue(boolean condition, String message) {
  if(!condition) {
    failNotEquals(condition, Boolean.TRUE, message);
  }
}
```

Рисунок 5.15 – assertTrue метод класа Assert

5.6 Написання автоматизованих тестів

Для того, щоб перевірити функціонал експорту продукту, потрібен зареєстрований користувач. В кожному тесті буде реєструватися новий користувач для більшої візуалізації поведінки людини.

Тому першим тестом доцільно буде вибрати процедуру реєстрації. Метод же після використовуватиметься у всіх тестах.

Типовий патерн для написання автотеста називається AAA:

- Arrange
- Act
- Assert

Приклад використання паттерну можна побачити на рисунку 5.17

```
[TestMethod]
public void AddTwoNumbers_Success()
{
    // Arrange
   const int firstNumber = 1;
   const int secondNumber = 2;
   var calculator = new Calculator();

    // Act
    var result = calculator.Add(firstNumber, secondNumber);

    //Assert
    Assert.AreEqual(3,result);
}
```

Рис 5.17 – Легкий приклад тесту калькулятора використовуючи паттерн AAA

У випадку з тестом на реєстрацію користувача:

- arrange - ініціалізація емейла і пароля

- act - клік на кнопку registration

-assert - перевірка, що користувач потрапив на сторінку MyDocs

В одному тестовому класі може бути багато тестів. Кожен тестовий метод позначається анотацією @Test. Існує величезна кількість корисних анотацій, які будуть зручні для написання того чи іншого тесту.

TestNG фреймворк запускає тести через .xml файл, в якому зібрана вся необхідна для нього інформація, приклад можна побачити на рисунку 5.18. У .xml можна виносити різноманітні параметри. Також існує велика варіація варіантів запуску тестів. Можна запустити весь клас тестів, можна окремо якийсь тест, можна один тест, але багато разів з різними параметрами.

```
<suite name="User Main Action Tests">

    <parameter name="email" value="oleksandr.biei@gmail.com"/>
    <parameter name="password" value="qwerty123"/>

    <test name="User Registration Test">
        <classes>
            <class name="tests.pdffiller.RegistrationTest"/>
        </classes>
    </test>

    <test name="User Login Test">
        <classes>
            <class name="tests.pdffiller.LoginTest"/>
        </classes>
    </test>

</suite>
```

Рисунок 5.18 – xml suite для регістрації та авторизації користувача системою

Натиснувши праву клавішу і вибравши Run Tests запуститься виконання цього пакету тестів.

Після реєстрації користувач потрапляє на сторінку MyDocs з обраним одним документом. Тобто він вже може робити перехід на сторінку експорту і перевіряти весь функціонал, що тестується в цій роботі системи.

Так як на сторінці Save As головний функціонал - це збереження файлу, то була необхідність перевіряти файл в скачується папці, а також перевіряти, що контент документа присутня.

Існує бібліотека PDFReadMan, яка вміє зчитувати pdf файл, використання якої наведено на рисунку 5.19. За допомогою неї можна:

- подивитись контент документа
- дізнатися кількість сторінок в документі

- відкрити документ з паролем

- перевірити кількість заповнюваних полів.

Якраз те, що потрібно для перевірки скачаного файлу з опціями PDFfiller.

```
saveAsExportPage.saveAs(ExportSaveAsData.PANEL_FORMAT_PDF, ExportSaveAsData.PANEL_DESTINATION_DESKTOP);

String downloadedFilePath = new File( pathname: TestData.PATH_TO_DOWNLOADS_FOLDER + "/" + filename + ".pdf").getAbsolutePath();
assertTrue(isFileDownloaded(downloadedFilePath),  s: "File was not downloaded");
assertEquals(PDFReadMan.getPagesNumber(downloadedFilePath), pages.length,  s: "Incorrect number of pages");
```

Рисунок 5.19 – Приклад перевірки кількості сторінок документу

Тести будуть писатися по вже готовим тест кейсів. У великих компаніях ручні тестувальники пишуть тест кейси, а потім передають їх автоматизованим тестувальникам, які перетворюють все написане в код.

5.7 Паралелізація тестів використовуючи фреймворк TestNG

Паралелізм, або многопоточность в термінах програмного забезпечення, це здатність ПЗ, операційної системи або програми виконувати безліч частин або компонентів іншої програми одночасно. TestNG дозволяє запускати тести в багатопотоковому режимі. Це означає, що на підставі конфігурації тестового набору, одночасно запускаються різні потоки, в яких виконуються тестові методи.

TestNG - це автоматизована система тестування з відкритим кодом; де NG означає NextGeneration. TestNG схожий на JUnit (особливо JUnit 4), але це не JUnit розширення. Натхненний JUnit. Він розроблений для кращого, ніж JUnit, особливо при тестуванні інтгрованих класів.

Ліквідація більшості обмежень старої системи, TestNG надає розробнику можливість писати більш гнучкі та потужні тести. Оскільки він сильно запозичує з Java Annotations (представлений з JDK 5.0) для визначення тестів, він також може показати вам, як використовувати цю нову функцію мови Java у реальному виробничому середовищі.

При виборі фреймворка для запуску тестів дуже важливо було знайти той, який може запускати тести паралельно в кілька потоків. Саме тому вибір впав

на TestNG, де це робиться дуже просто. При оголошенні сьюта .xml можна вказати, що саме буде паралелізоватись і в скільки потоків.

```
<suite name="My suite" parallel="methods" thread-count="5">
<suite name="My suite" parallel="tests" thread-count="5">
<suite name="My suite" parallel="classes" thread-count="5">
<suite name="My suite" parallel="instances" thread-count="5">
```

Рисунок 5.20 – Приклад оголошення паралелізації з документації TestNG

Як видно з рисунку 5.20 - можна паралелити методи, тести, класи. У дипломній роботі буде показаний приклад паралеллізації тестів.

5.8 Налаштування та запуск тестів

5.8.1. Настройка Maven

Одним словом, Maven допомагає розробникам керувати всіма залежностями проекту та забезпечувати легкий процес збирання. Крім того, Gradle є розширенням Maven і працює надзвичайно добре. Нові розробники, як правило, віддають перевагу Граблі над Maven

Цілі Maven включають в себе:

- Забезпечення єдиної системи збірки
- Надання якісної інформації про проект
- Надання рекомендацій щодо розробки найкращих практик
- Дозвіл прозорої міграції на нові функції

Рис 5.21 – Деякі інструменти для тестування в pom.xml

Дуже важливим кроком для поставленої мети є правильна настройка pom.xml файлу, в якому вказані всі залежності. При новому проекті створюється pom.xml файл, проте в нього потрібно ще багато всього додати. Невід'ємними частинами є інструменти для тестування, які наведені рисунку 5.21. В налаштуванні проекту можна побачити Maven вкладку, де є багато можливостей для покращення роботи. Рисунок 5.22 показує цю вкладку.

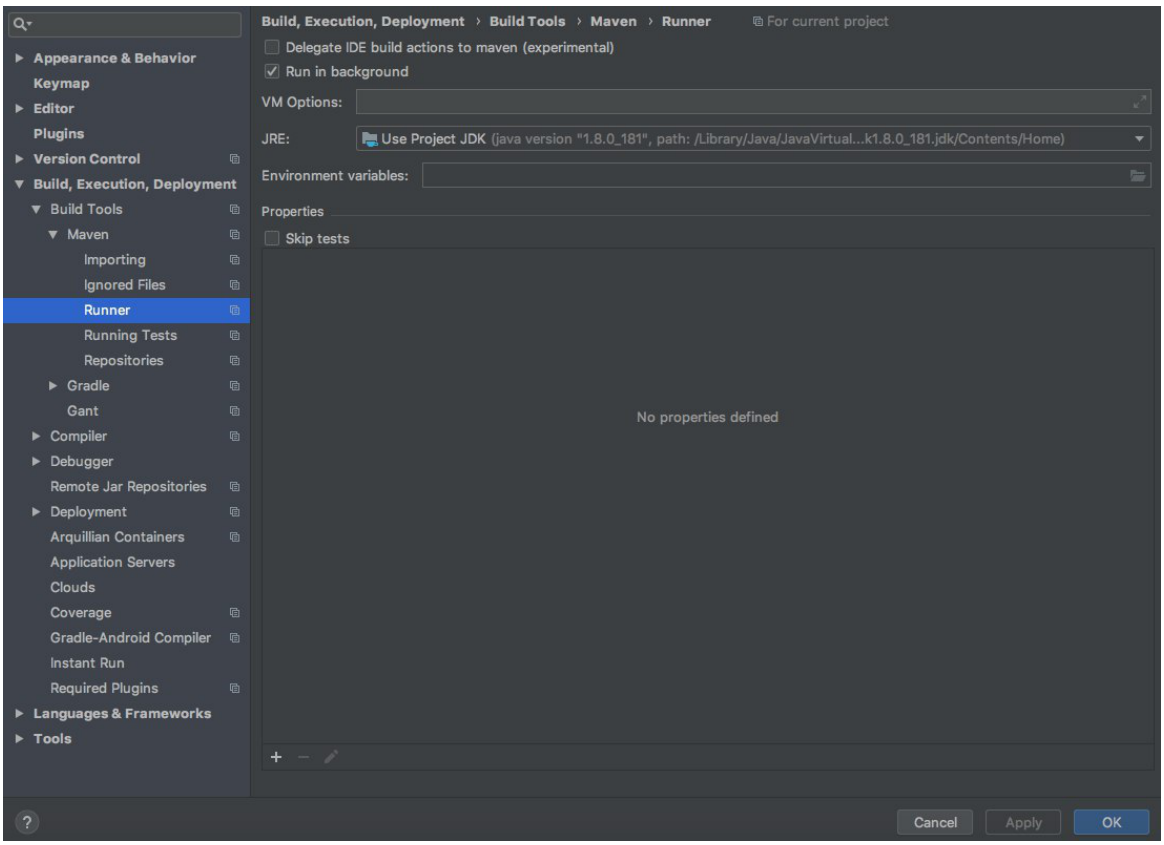


Рис 5.22 – Налаштування Maven у проекті

Основні можливості Apache Maven:

- Це інструмент для створення
- Спрощує процес збірки
- Вирішує залежності
- Приймає турботи з перевірки на встановлення / розгортання -> життєвий цикл Maven
- Пакети будуються як JAR / WAR / EAR -> стандартна упаковка Java / JEE
- Працює модульні тести та інтеграційні тести
- Проблеми без Maven (тобто переваги використання Maven)
- Генерує документи (наприклад, javadoc) з вихідного коду
- Допомагає створювати сайти, звітування та документацію
- Дотримується стандартна структура проекту (або структура папок)

5.8.2. Установка і настройка Jenkins

Насамперед варто завантажити і встановити Jenkins. Після завершення встановлення можна перейти до інформаційної панелі Jenkins (http://localhost: 8080 за замовчуванням) у вікні веб-переглядача.

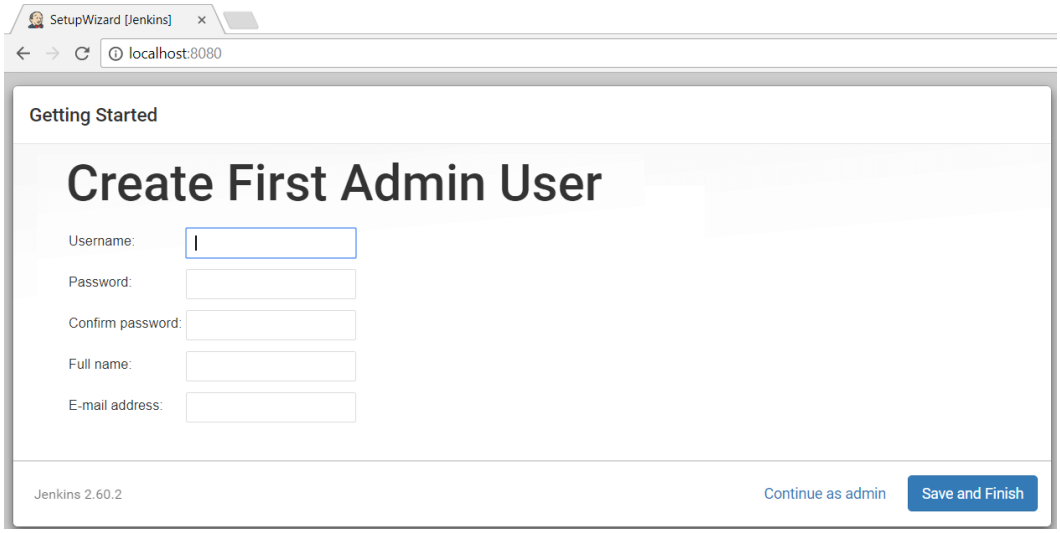


Рисунок 5.23 – Створення юзера в Jenkins

Для того, щоб працювати з системою, потрібен аккаунт. Адміністратор має можливість створювати користувачів системи та обмежувати їх у діях. На рисунку 5.23 можна побачити інформаційну панель при створенні користувача. У системі задіяно три типи юзерів:

- Тестувальник
- Автоматизований тестувальник
- Адміністратор

У додатку можна знайти діаграму використання, де це написано, хто які дії може виконувати.

Після створення користувача можна приступити до додавання Джоб (завдання, які будуть виконуватися). Натиснувши New Item (рисунок 5.24) йде перехід на нове завдання, де потрібно правильно налаштувати параметри

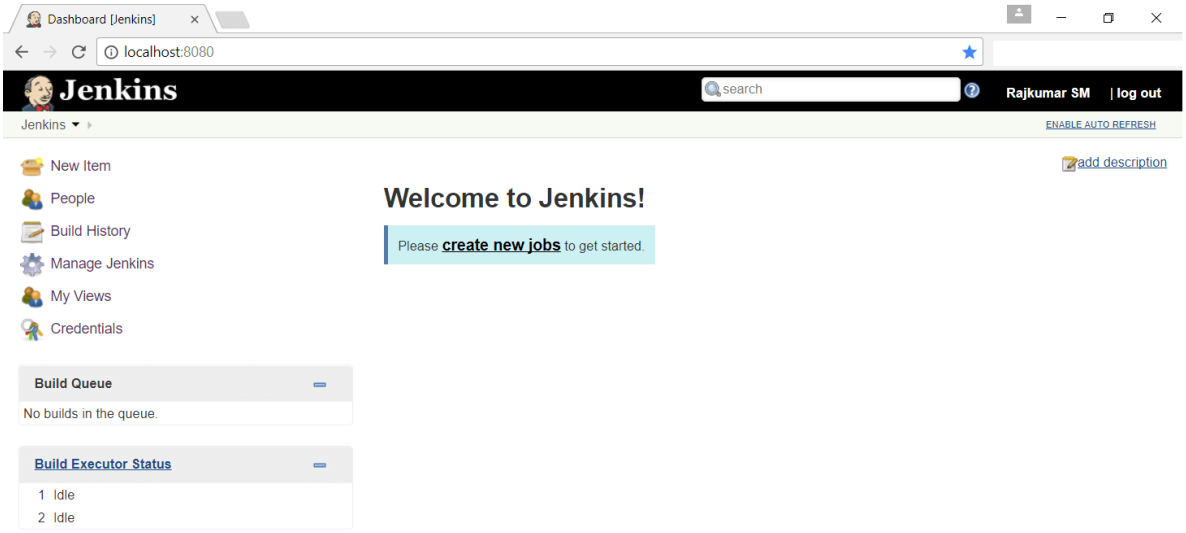


Рисунок 5.24 – Головна сторінка Jenkins

Назва для тестової збірки - TestNGProject. Натиснувши на пункт 'TestNGProject' на вкладці «Загальні» можна побачити «TestNGProject». Кнопка "Додатково" є для додавання робочої області. На рисунку 5.25 видаляється прапорець Use Custom Worspace та вводиться шлях до робочого простору.

Також треба підв'язати git аккаунт до налаштування джоби, а також вказати сервер, на якому джоба буде запускатись.

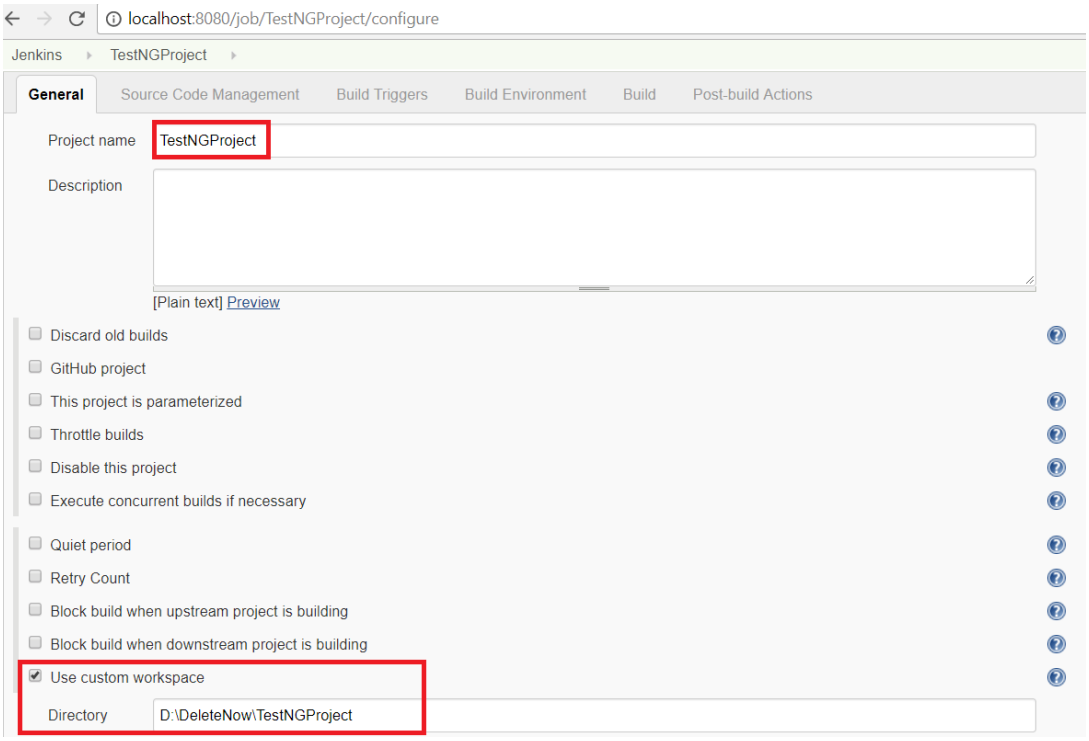


Рисунок 5.25 – Конфігурація задачі

Щоб прив'язати сервер, на якому будуть виконуватися завдання Jenkins, в настройках варто вибрати "Add new Server" і завантажити .jar файл, який після потрібно встановити на серверному комп'ютері. До того ж, необхідно встановити на серверному комп'ютері git, адже файли будуть автоматично стягуватися з віддаленого git сховища. Ніяких бібліотек встановлювати не варто, все буде в тестовому фреймворку.

Запустивши проект, спочатку Jenkins піде на сервер, де знайде .pom файл, в якому знаходяться всі залежності тестового фреймворка. Назва .xml передається з Jenkins настройки завдання, а потім при запуску додається в maven .pom файл, а саме в "suiteXmlFiles", як на рисунку 5.26. Після запускається TestNG, а після драйвер браузера. Як працює система можна подивитись на діаграмі послідовності, яка знаходиться у додатку Б.

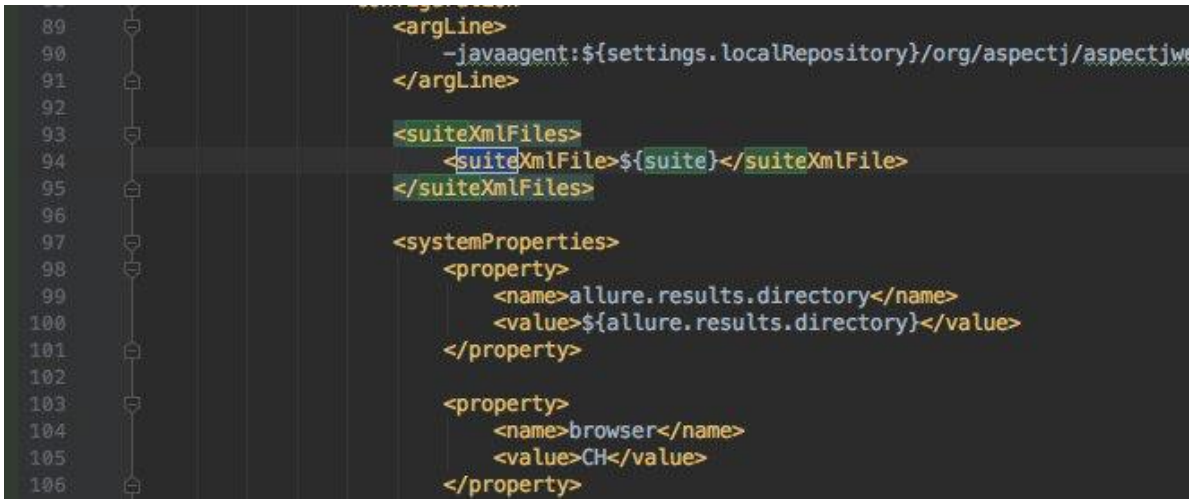


Рисунок 5.26 – Місце, куди параметром передається назва зборки

Запустивши перший раз та натиснувши консольний вивід можна побачити результат, який зображений на рисунку 5.27. При кліку на кнопку Run, Jenkins буде викликати testng.xml з пакетного файлу.

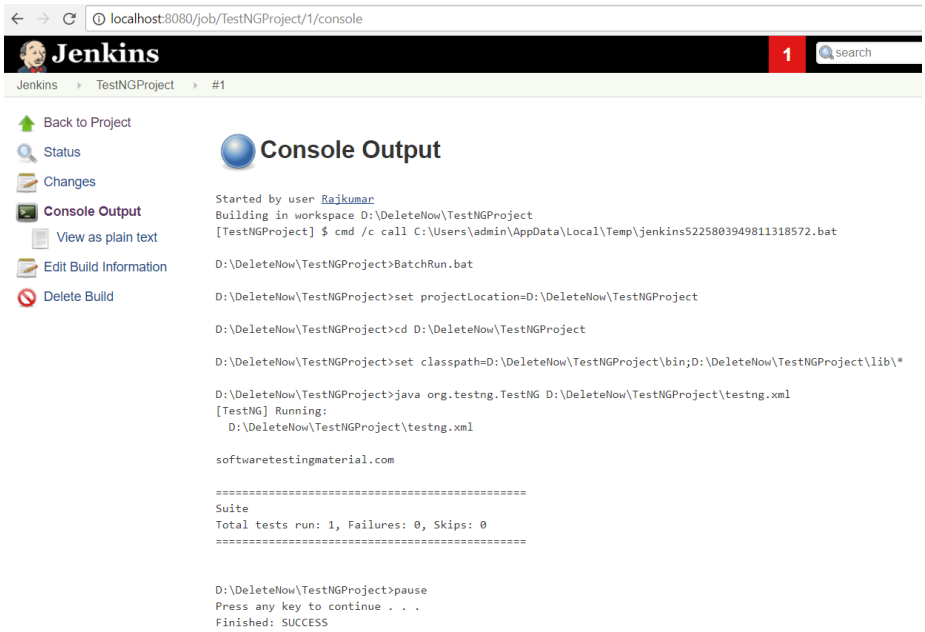


Рис 5.27 - Консольний вивід першого білда

Висновки з розділу

Результатом розробки стала система автоматизації процесів тестування програмного забезпечення з використанням паралелізації тестів , що включає у себе фреймворк для автоматизації тестування з написаними тестами та описаними сторінками тестованого продукту, настроєний Jenkins та сервер для прогону тестів.

При створенні системи досліджено переваги та недоліки вже існуючих на ринку систем тестування програмного забезпечення розглянутих у розділі 3. Були розглянуті основні інструменти для автоматизованого тестування та обґрунтований вибір вибраних інструментів. Була налагоджена паралелізація тестів з прикладами реалізації. Розроблена система відповідає критеріям якості, що були висунуті у розділі 2 та технічному завданню магістерської дисертації.

Система автоматизації процесів тестування програмного забезпечення, що була створена у магістерський дисертації має на меті створення системи для автоматичної перевірки функціоналу тестованого об'єкта зі зручним інтерфейсом виводу помилок.

Варто зауважити що система загалом є сучасною дружньою для користувача системою і може бути впроваджена як у бізнес-процеси

тестувального відділу великих компаній, що працюють із великим набором функціоналу, так і використовуватися в маленьких компаніях для підвищення якості виробленого продукту.

6 ТЕСТУВАННЯ СИСТЕМИ

6.1 Прогін тестів та аналіз

Написавши і згрупувавши всі, вийшло 7 пакетів тестів, в кожному з яких від 2 до 5-ти тестів. Запустивши тест через певний час ми отримаємо результат. Так як продукт працює стабільно, то тести будуть теж стабільно працювати. Далі буде наведено приклад прогону пакета тестів на збереження документа в різні формати на рисунку 6.1.

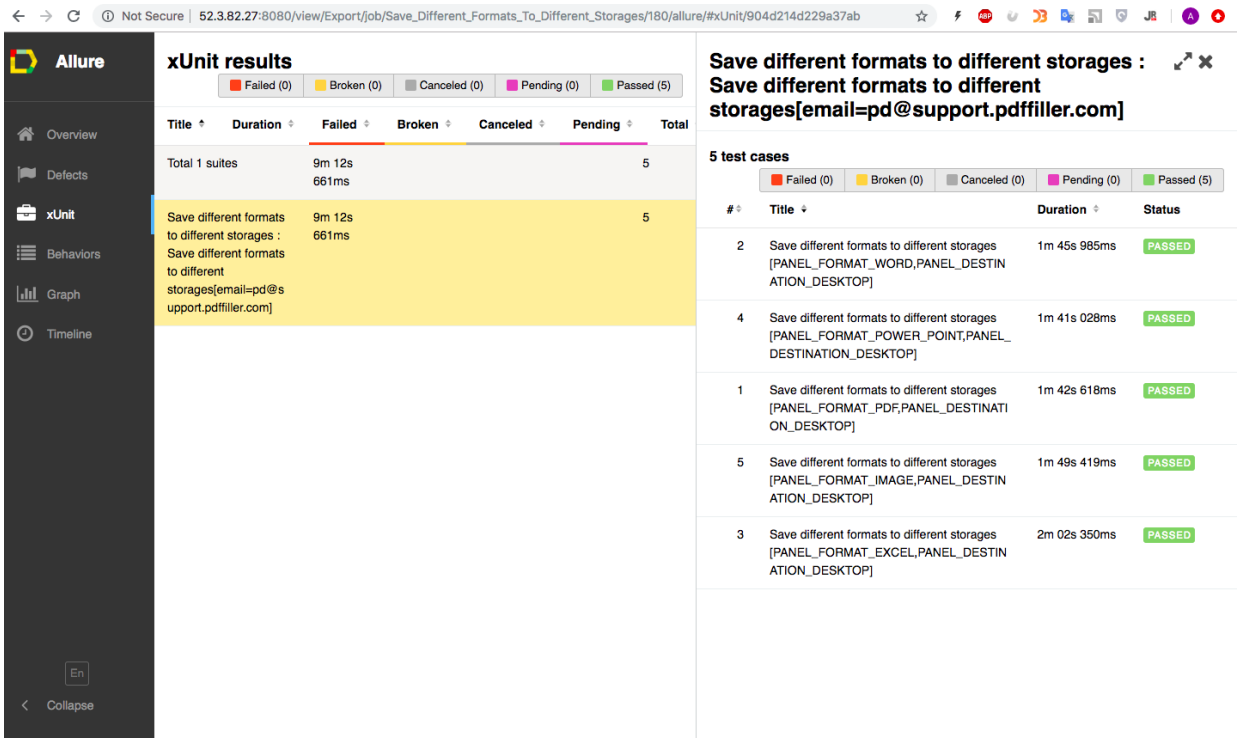


Рисунок 6.1 – Виведення результатів пакету тестів на збереження файлу

Із звіту Allure видно, що час прогону всього пакету зайняло 9 хвилин. Тут тест запускався з одного потоку. Натиснувши на назву тесту можна побачити кроки, які виконувалися, щоб перевірити функціонал.

Формат збереження і шлях збереження є параметрами для тесту. Вони винесені в окремий enum для зручності.

```
public enum ExportSaveAsData {

    // -- FORMATS

    PANEL_FORMAT_PDF("PDF"),
    PANEL_FORMAT_WORD("Word"),
    PANEL_FORMAT_EXCEL("Excel"),
    PANEL_FORMAT_POWER_POINT("PowerPoint"),
    PANEL_FORMAT_IMAGE("Image"),
```

Змінивши .xml файл і поставивши 2 потоки можна помітити менший час прогону всього пакету тестів. А коли сервер з хорошими характеристиками, то можна запускати тести і в 3-4 потоки.

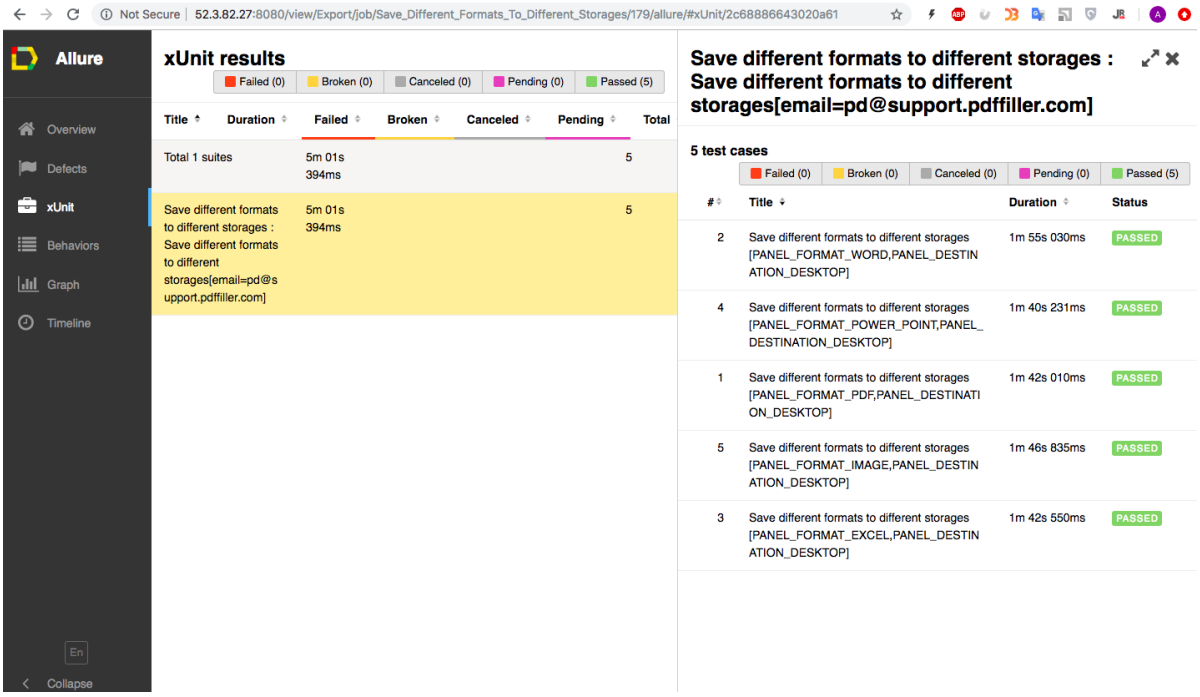


Рисунок 6.1 – Виведення результатів пакету тестів на збереження файлу у 2 потоки

Таблиця 6.1 – Порівняння часу прогону тестів в один та 2 потоки

№	Suite	Tests	Час прогону	Час прогону з паралелізаією
1	Авторизація, реєстрація	1. Авторизація 2. Реєстрація	3 хв 40 секунд	2 хвилини

2	Перехід на експорт з файлами	1. 1 файл 2. 2 файли 3. 3 файли	3 хвилини	2 хвилини
3	Збереження файлу у різні формати	1. Pdf 2. Excel 3. PowerPoint 4. Word 5. Image	9 хвилин	5.5 хвилин
4	Збереження з опціями	1. Fillable fields 2. Password 3. Add pdffiller to name	7 хв 30 сек	5 хвилин
5	Збереження з unselect pages	1. Unselect 2 pages 2. Unselect all pages	3 хв 30 сек	2 хв
6	Мультізбереження	1. Збереження двух файлів 2. Збереження 3 файлів	4 хв	2 хвилини 30 секунд
7	Платний документ	1. Перехід на експорт з платним документом	1 хвилина	1 хвилина

6.2 Тест з помилкою

Є тест на перевірку збереження fillable полів, код якого наведений на рисунку 6.3 Залишимо перевірку, але збережемо документ без цієї опції.

```
String downloadedFilePath = new File( pathname: TestData.PATH_TO_DOWNLOADS_FOLDER + "/" + projectName + ".pdf").getAbsolutePath();
assertTrue(isFileDownloaded(downloadedFilePath),  s: "File was not downloaded");
String[] fillableFields = PDFReadMan.getPageFillableFields(downloadedFilePath,  page: 2);
assertEquals(fillableFields.length,  i1: 2,  s: "Fillable fields isn't correct");
```

Рисунок 6.3 – Перевірка збереження fillable полів

Якщо кількість заповнюваних полів не буде рівних двом, то буде помилка "Fillable fields is not correct" та тест буде Failed, тобто зламаний саме тестований функціонал., що можна побачити на рисунку 6.5. Якби якийсь з шагів до доступу к цьому функціоналу зламався, то тест був би зі статусом Broken, про що і йшла мова у розділі про написання фреймворку для тестування.

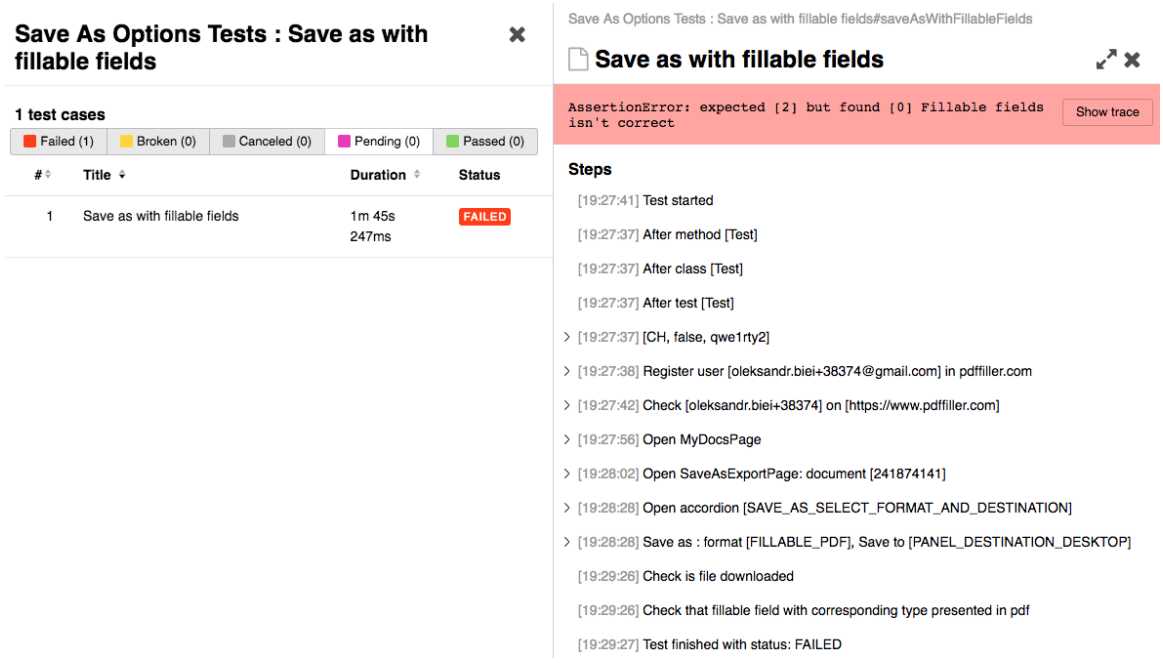


Рисунок 6.4 – Тест з помилкою тестованого функціоналу

Висновки з розділу

Було протестовано розроблену систему автоматизації процесів тестування програмного забезпечення з використанням паралелізації тестів на прикладі

вибраного об’єкту для тестування – сайт PDFfiller, а саме функціонал експорту Save As. Результати показали, що паралелізований запуск тестів значно заощаджує час на тестування. Також був протестований негативний кейс, коли розроблена система знаходить помилки та вказує на помилку.

7 РОЗРОБЛЕННЯ СТАРТАП-ПРОЕКТУ

7.1. Опис ідеї проекту (товару, послуги, технології)

Технологія відноситься до засобів обробки та аналізу інформації, призначена для компаній, націлених на розвиток напрямку тестування, автоматизації процесу тестування і підвищення контролю якості свого продукту. Система показує ефективність впровадження автоматизованого тестування.

Тестування продукту, використовуючи цю систему, займає набагато менше часу ніж міг би на нього витратити ручної тестувальник.

Таблиця 5.1 - Опис ідеї стартап-проекту

Описання ідеї	Варіанти застосування	Плюси для споживача
	1.Тестування	Немає готових аналогів подібної системи. Економія часу на тестування та підвищення якості тестування.
	2.Наукова діяльність	Застосування новітніх технологій.

Застосування сучасних технологій та застосування алгоритмів автоматизації робить програмне забезпечення конкурентоспроможним. Відомо, що в Україні мало хто використовує подібну схему для автоматизації

тестування, отже наявність вітчизняного алгоритму для перевірки якості є дуже перспективним.

Таблиця 5.2 - Визначення сильних, слабких та нейтральних характеристик ідеї проекту.

№ п/п	Техніко-економічні характеристики ідеї	(потенційні) продукти/алгоритми конкурентів				W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
		Мій проект	Автоматизація з TestComplete	Автоматизація з Watir	Автоматизація з Robot framework			
1.	Вартість ПЗ	Низька	Висока	Висока	Низька			+
2.	Час обробки	Низький	Високий	Високий	Високий		+	
3.	Автоматизація	80 %	60%	70%	60%		+	
5.	Споживачі (відомий бренд)	-	-	+	+	+		

7.2. Технологічний аудит ідеї проекту

Таблиця 5.3 - Технологічна здійсненність ідеї проекту

№ п/п	Проектна ідея	Реалізація та технології	Наявність технологій	Технологічна досупність
1.	Перевірка функціоналу повторюючи дії користувача у браузері	Функції Selenium дозволяють автоматизувати дії	Наявні.	Так.
2.	Автоматизація процесу	TestNG+Jenkins	Наявні.	Так.

7.3. Аналіз ринкових можливостей запуску стартап-проекту

Найголовнішою характеристикою даного продукту є економія часу та економія робочої сили. Завдяки цьому продукту можна замінити деяких ручних тестувальників. Тому, зависока ціна іноземних аналогів та їх недоступність для широкого кола споживачів робить даний алгоритм перспективним для застосування як в Україні, так і закордоном.

Таблиця 5.4 - Попередня характеристика потенційного ринку стартап-проекту

№ п/п	Стан ринку (показники)	Характеристика

1	Кількість головних гравців, од.	0. Тільки закордонні, немає аналогів в Україні.
2	Загальний обсяг продаж, ум.од/час	Поодиничний продаж, закордонних товарів.
3	Якісна оцінка динаміки ринку	Зростає
4	Наявність обмежень для входу (вказати характер обмежень).	Новітня технологія потребує ресурсів, практичної перевірки та доробки універсального алгоритму.
5	Специфічні вимоги до стандартизації та сертифікації.	Потребує медичної сертифікації.
6	Середня норма рентабельності в галузі (по ринку), %	50

Основною споживчою аудиторією є ІТ компанії, які дбають про якість свого продукту. Так як персонал самостійно не може купувати даний алгоритм, розповсюдження ПЗ відбувається опосередковано через дистриб'юторів/ представників. Алгоритм є універсальним, без залежності від рекомендацій установи та роздільної здатності зображення.

Таблиця 7.5 - Характеристика потенційних клієнтів стартап-проекту

№ п/п	Риночна потреба	Аудиторія	Відмінності у поведінці різних

			потенційних цільових груп клієнтів
1. Дешевизна алгоритму		Зацікавлені в купівлі дешевшого.	
2. Зменшення ризику виникнення людського фактору.		Зацікавлені в кращій якості.	
3. Купувати вітчизняне		Зацікавлені в підтримці національного виробника.	
4.Отримати закордонний аналог по якості за українською ціною		Зацікавлені за менші гроші отримати високоякісний матеріал.	

Основною загрозою є конкурентоспроможність – в Україні застосовували тільки закордонні аналоги. Але завдяки рекламі можна вивести цей продукт на міжнародний рівень.

Таблиця 7.5 - Фактори загроз

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
1.	Конкурентоспроможність	Новий товар на ринку-невідомий бренд-низька конкурентоспроможність	Реклама, вдалий маркетинговий проект, залучення дилерів, спонсорів до співпраці.
2.	Невідомий бренд	Невідомий новий товар невідомої фірми.	Розкрутка товару, розповсюдження серед лікарів і в наукових центрах.
3.	Комерційна таємниця	Можливість відкриття технології невідомими робітниками.	Підписання строгих контрактів з несенням матеріальної компенсації.

Найголовнішою можливістю є залучення іноземних спонсорів та можливостей реалізації товару закордоном.

Таблиця 7.6. - Фактори можливостей

№ п/п	Фактор	Зміст можливості	Можлива реакція компанії

1.	Залучення іноземних спонсорів.	Залучення іноземних капіталів, підтримка спонсорів.	Укладення договорів про співпрацю та налагодження торгових контактів.
2.	Налагодження зв’язків з CEO великих продуктових IT компаній	Залучення дистриб’юторів та наукових представників.	Пошук та налагодження зв’язків з компаніями.

Таблиця 7.7 - Ступеневий аналіз конкуренції на ринку

Характеристика конкурентного середовища	В чому проявляється дана особливість	Можливі дії компанії, щоб бути на рівні
1. Вказати тип конкуренції - монополія/олігополія/ монополістична/чиста	Монополія, на даний час в Україні немає аналогів.	Розкрутка національного товару, залучення інвесторів.
2. За рівнем конкурентної боротьби	Міжнаціональний	Випуск аналогу закордонних продуктів, дешевша ціна на аналогічну продукцію.
3. За галузевою ознакою	Внутрішньогалузева	Використовується тільки в медицині в якості штучного

		аналогу натуральних судин.
4. Конкуренція за видами товарів	Товарно-родова	Власні розробки, унікальна технологія, реклама.
5. За характером конкурентних переваг	Цінова	Ціна нижча за іноземні аналоги.
6. За інтенсивністю - марочна/не марочна	Не марочна	Реклама, покращення технологій

Таблиця 7.8 - Аналіз конкуренції в галузі за М. Портером

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
	Немає, або закордонні	Гнучкі ціни,розмір капіталовкладень	Велика концентрація постачальників	Якісна продукція, не потребує великої кількості товару	Вища ціна, вищі змінні витрати
Висновки	Невисока конкурентна боротьба	Є можливості входу в ринок, наявні потенційні конкуренти	Постачальники диктують умови роботи на ринку, наприклад, ціну та швидкість	Так, залежно від попиту на товар.	Обмежень немає

			розповсюдження		

Робота на ринку є можливою, залежить від купівельної спроможності установ. Так як вони купують набагато дорожчий товар закордонних виробників, можна зробити висновок про те, що даний товар буде користуватись періодичним попитом. Також, важливим є фактор розповсюдження шляхом постачальників. Конкурентна боротьба є високою, проте лише з закордонними аналогами, адже таких в Україні немає.

Таблиця 7.9 - Обґрунтування факторів конкурентоспроможності

№ п/п	Конкурентоспроможність	Обгрунтуваннння
1	Час	Менша необхідність машинної праці та економія часу.
2	Ціна	Ґрунтується на собівартості-отже є нижчою.
3	Якість	Не гірша за закордонні аналоги.

Таблиця 7.10 - Порівняльний аналіз сильних та слабких сторін власного проекту

№ п/п	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні з власною компанією						
			–3	–2	–1	0	+1	+2	+3
1	Час	15				+			
2	Ціна	18	+						
3	Якість	14						+	

Таблиця 7.11 - SWOT- аналіз стартап-проекту

Сильні сторони:	Слабкі сторони:

Собівартість Ціна Рентабельність	Невідомий бренд Залежність від інших сервісів Потрібно встановити додаткове ПО
Можливості: Вихід на світовий ринок Забезпечення споживчих потреб Дохід	Загрози: Допоміжний сервіс вийде зі строю Недостатня реалізація

Таблиця 7.12 - Альтернативи ринкового впровадження стартап-проекту

№ п/п	Варіанти поведінки на ринку	Шанс на отримання ресурсів	Дедлайн
1	Практичне використання алгоритму,	+	2 роки
2	вдосконалення	-	1 рік
3	Залучення іноземних фахівців	+	0,5 року
4	Рекламна кампанія	+	5 років
5	Отримання міжнародних сертифікатів Вихід на закордонний ринок	+	6 років

7.4. Розроблення ринкової стратегії проекту

Таблиця 7.13 - Вибір цільових груп потенційних споживачів

№ п/п	Описання профілю групи майбутніх клієнтів	Прагнення покупців здійснити результат	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції	Простота входу у сегмент
1 2	Інформаційні технології(ІТ компанії) Наукові центри	Не готові Готові	Дуже необхідно Дуже необхідно	Мінімальна Велика	Важка Легка
Які цільові групи обрано: ІТ компанії					

Таблиця 7.14 - Визначення базової стратегії розвитку

№ п/п	Обрана альтернатива розвитку проекту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку
1	Інформаційні технології	Стратегія лідерства по витратах	- ціна - простота застосування	Детальний нагляд за стабільними витратами, скорочення виробничих,

			- продуктивність роботи - низькі витрати з розрахунку на кількість пацієнтів на рік	збутових і маркетингових витрат, здійснення капіталовкладення, зосередженої в зниження витрат

Таблиця 7.15 - Визначення базової стратегії конкурентної поведінки

№ п/п	Чи є проект «першопрохідцем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки
1	Так	Буде шукати	Ні, продукт застосовує власний алгоритм для ПЗ.	Стратегія наслідування лідера

Таблиця 7.16 - Визначення стратегії позиціонування

№ п/п	Умови до товару цілеспрямованій аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проекту	Вибір асоціацій для формування позиції свого проекту
1	Дешевизна закупівлі підтримки ПЗ.			
2	Зменшення ризику людської помилки.			
3	Купувати вітчизняне.			
4	Отримати закордонний аналог по якості за українською ціною.			

7.5. Розроблення маркетингової програми стартап-проекту

Таблиця 7.17 - Визначення ключових переваг концепції потенційного товару

№п/п	Потреба	Вигода, яку пропонує товар	Переваги перед конкурентами (існуючі або такі,

			що потрібно створити)
1	Дешевизна ПЗ	Низька ціна	Ціна
2	Купувати вітчизняне	Національний продукт	Якість
3	Отримати закордонний аналог по якості за українською ціною	Національний продукт	Універсальність

Таблиця 7. 18. - Опис трьох рівнів моделі товару

Рівні товару	Сутність та складові		
І. Товар за задумом	Система автоматизації процесів тестування програмного забезпечення прискорює роботу відділу тестування в великих компаніях.		
	Властивості/характеристики	М/Нм	Вр/Тх /Тл/Е/Ор
	1. Висока швидкість 2. Якість тестування 3. Сумісний з більшістю ПК		
	Якість: стандарти, нормативи, параметри тестування тощо Стандартизація відповідно до ДСТУ,ISO. Регламентується НД, СРМ.		
	Пакування відсутнє.		
	Марка: назва організації-розробника + назва товару		
Потенційний товар буде захищено від копіювання: патентування, сертифікати відповідності.			

Таблиця 7.20 - Формування системи збуту

№ п/п	Особливість купівельні дії цільових покупців	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
	Придбання імовірна уже після інструктажу і Демонстрації ЗГІДНО, з детальним поясненням імовірний і обмежень проекти.	Маркетингові дослідження, сервіс реалізованих товарів, твердження в собі карб торговельних договорів	Канал нульового рівня (прямий маркетинг)	Торгівля через веб-сайт, що належить виробнику.

Таблиця 7.21 - Концепція маркетингових комунікацій

№ п/п	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення

1	Довіра до тестувальників, які розхвалюють товар	Державні інститути, наукові установи	Налагодження контактів з відомими IT компаніями	Донести основну ідею, цінову політику, якість	Продемонструвати переваги перед існуючим товарами

Висновки з розділу

У цьому розділі було проведено аналіз розробленої системи автоматизації процесів тестування програмного забезпечення з використанням паралелізації тестів у якості стартап-проекту. Варто зауважити, що проект має можливість ринкової комерціалізації, через те, що ринок систем перевірки якості програмного забезпечення потребує якісного та інноваційного продукту для автоматизації тестів.

Результатом роботи є розроблений стартап-проект, план виходу на ринки програмного забезпечення та маркетингова стратегія. Створений стартап-проект доцільно застосувати при комерціалізації розробки.

ВИСНОВКИ ДО МАГІСТЕРЬСКОЇ ДИСЕРТАЦІЇ

В роботі була проаналізована сьогоднішня ситуація у сфері тестування програмного забезпечення. Було розглянуто і проаналізовано основні сучасні методи та види тестування програмного забезпечення, як ручного, так і автоматизованого. Виявлені переваги і недоліки кожного з методів і запропоновано метод, що поєднує ці методи в одне ціле. Система показує, що

тільки з автоматизованим тестуванням функціонал не буде завжди на високому рівні, потрібен ручний тестувальник для моніторингу ситуації.

Було вибрано об’єкт для тестування, який задовольняв вимогам для реалізації системи. Було вибрано оптимальну кількість тестів для перевірки обраного функціоналу та написано під цей функціонал тест кейси.

Також було виявлено переваги створення власного фреймворку для тестування обраного продукту. Був розроблений фреймворк виходячи з аналізу тестованого об’єкту та аналізу інструментів для тестування. Було описано і вибрано архітектуру фреймворку . Після цього було обрано набір інструментів для вирішення поставлених задач.

Були написані автоматизовані тести з використанням фреймворку для тестування. Було обрано архітектуру тестів, яка найбільш доцільна для вирішення поставленої задачі — автоматизація тестів з використанням паралелізації.

Була налаштована система запуску автоматизованих тестів на основі проаналізованих даних. Використані плагіни для кращого відображення результатів тестування. Була введена паралелізація до запуску тестів, протестовано та проаналізовано.

Розроблену систему було протестовано у розрізі негативних та позитивних кейсів. За результатами тестування стало зрозуміло, що система значно заощаджує час на тестування продукту та робить процес тестування якіснішим.

На основі обраних методологій і інструментів, враховуючи користувацькі вимоги було розроблено систему автоматизації процесів тестування програмного забезпечення. Розроблена система виконує поставлені задачі і задовольняє поставленим вимогам.

Було розроблено стартап проект і обраховано перспективи розвитку розробленої системи як стартап-проекту та шляхи її монетизації. За результатами дослідження було зроблено висновок що розроблену систему доцільно розвивати як стартап проект.

Подальші дослідження доцільно спрямувати на розширення середовищ для тестування, розширення видів тестування, оптимізацію алгоритмів початку зборки для тестування.

СПИСОК ПОСИЛАНЬ

1. Библиотека MSDN. Источник информации для разработчиков, использующих средства, продукты, технологии и службы корпорации Майкрософт. [Электронный ресурс].
   Режим доступа: http://msdn.microsoft.com
2. Савин Р. Тестирование Дот Ком, или Пособие по жесткому обращению с багами в интернет- стартапах. - М.: Дело, 2007. - 312с
3. Тестирование и качество ПО. [Электронный ресурс].
   Режим доступа: http://software-testing.ru/
4. SeleniumHQ Browser Automation. [Электронный ресурс]. –
   Режим доступа: http://docs.seleniumhq.org
5. Whittaker J. How to Break Web Software / James Whittaker., 2003. – 119 с.

6. Kaner C. Testing Computer Software, 2nd Edition / C. Kaner, H. Q. Nguyen, J. Falk,., 1999. – 496 c. – (ISBN: 978-0-471-35846-6)
7. Do good code: 8 правил хорошего кода [Електронний ресурс] // GeekBrains Режим доступу до ресурсу: https://habr.com/company/geekbrains/blog/270001/.
8. Black R. Critical Testing Processes. Plan, Prepare, Perform, Perfect / Rex Black., 2011. – 544 c.
9. M. Kirund, H. S. Lopes, T. Pret, Big company testing software, Sci. Rep. 3.
10. Groff J. SQL, the complete reference / James R Groff., 1999.
11. Lerman J. Programming Java. — 2nd Edition. / Julia Lerman., 2010. – 920 c.
12. Отношения классов - от UML к коду [Електронний ресурс] Режим доступу до ресурсу: https://habrahabr.ru/post/150041/.
13. Pro Git book [Електронний ресурс] // 1st Edition. – 2009. Режим доступу до ресурсу: https://git-scm.com/book/ru/v1
14. Diabetes-Induced Cardiomyocyte Passive Stiffening Is Caused by Impaired Insulin-Dependent Titin Modification and Can Be Modulated by Neuregulin-1. / Hopf AE, Andresen C, Kötter S та ін.]. // Circulation research. – 2018. – С. 342–355.
15. D. Dikaiakos M. Distributed Internet Computing for IT and Scientific Research [Електронний ресурс] / Marios D. Dikaiakos – Режим доступу до ресурсу: https://www.computer.org/csdl/mags/ic/2009/05/mic2009050010.pdf.
16. S. Manvi S. Resource management for Infrastructure as a Service (IaaS) in cloud computing [Електронний ресурс] / Sunilkumar S. Manvi – Режим доступу до ресурсу: http://tarjomefa.com/wp-content/uploads/2017/10/TarjomeFa-F148-English.pdf.
17. Rouse M. Software as a Service (SaaS) [Електронний ресурс] / Margaret Rouse – Режим доступу до ресурсу: https://searchcloudcomputing.techtarget.com/definition/Software-as-a-Service.
18. Джошуа Б. Java. Эффективное программирование / Блох Джошуа., 2014.

19. Хорстманн К. Java. Библиотека профессионала. Том 1. Основы / Хорстманн К, Корнелл Г., 2016. – 866 с. – (10).

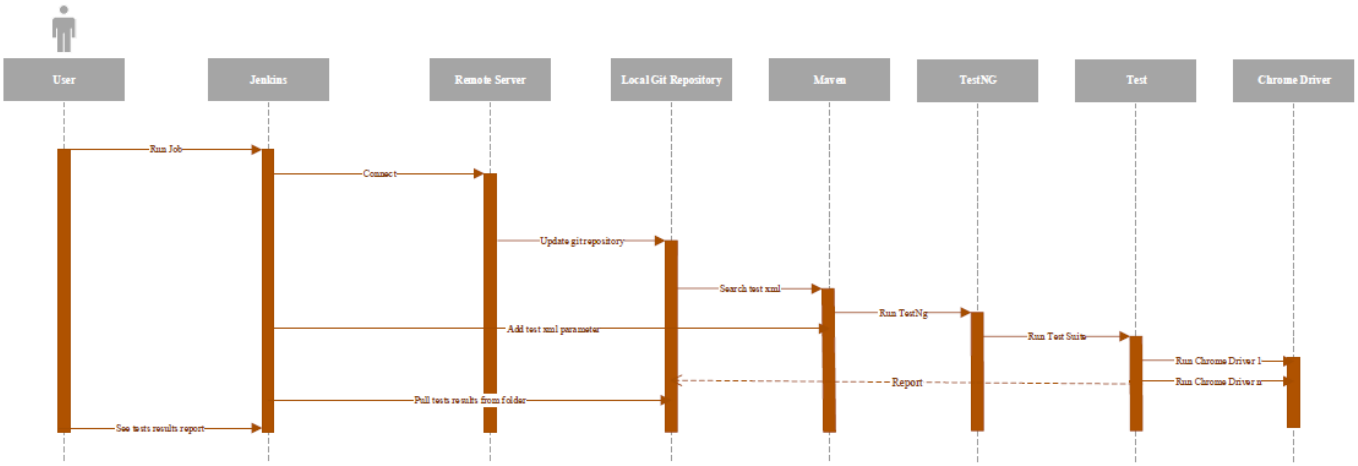
20. Q. Li, Y. Chen, L. L. Jiang, P. Li, H. Chen, A tensor-based information framework for predicting the stock market, ACM Trans. Inform. Syst. (TOIS) 34 (2) (2016) 11.

ДОДАТОК А

Діаграма варіантів використання

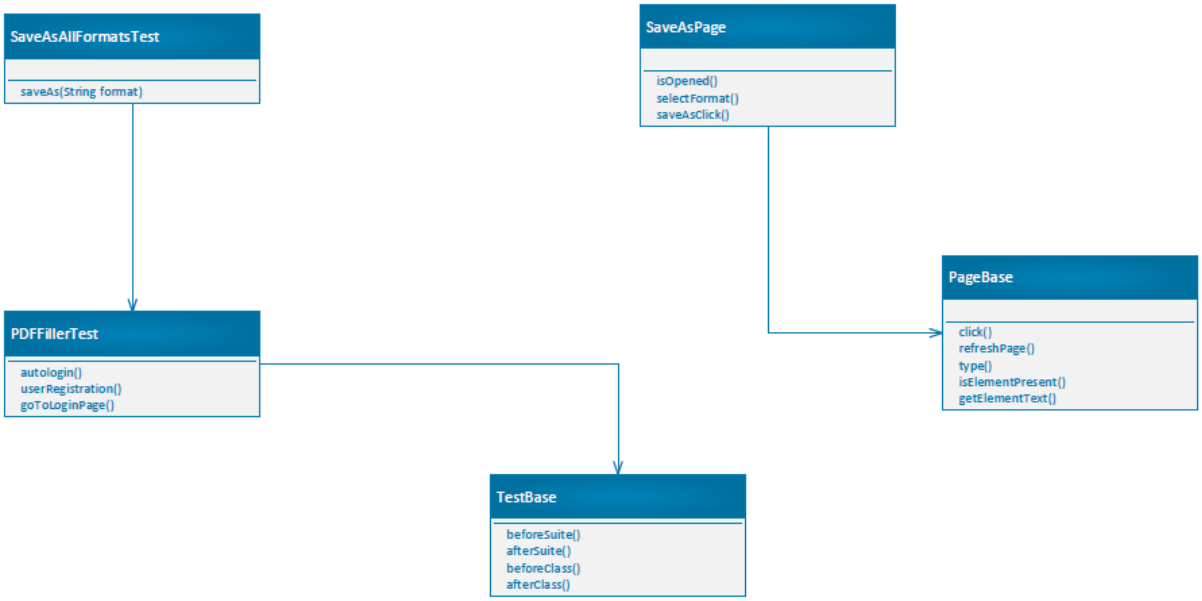
ДОДАТОК Б

Діаграма послідовності



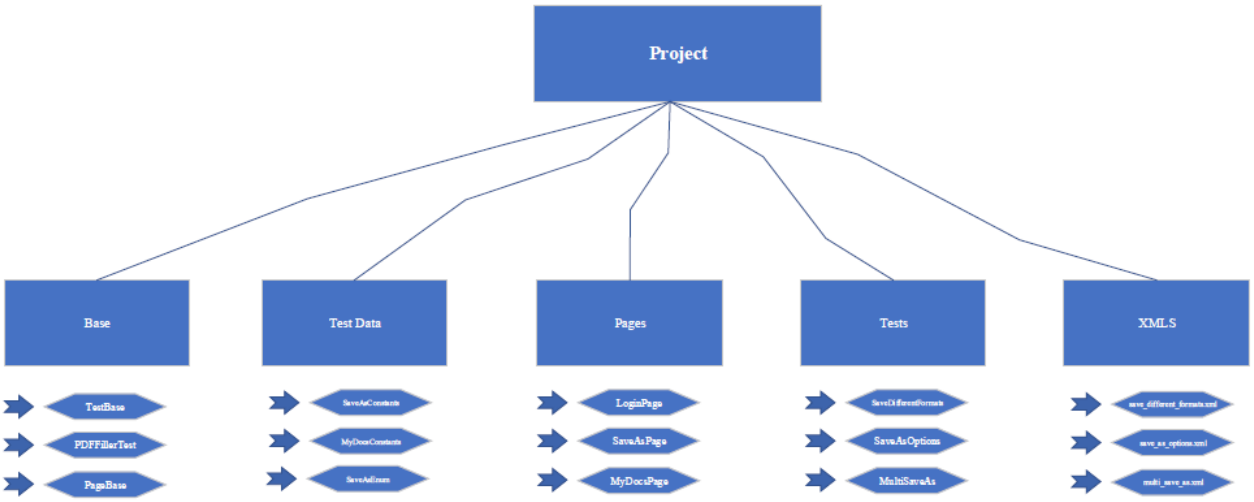
ДОДАТОК В

Діаграма класів фреймворка для автоматизації



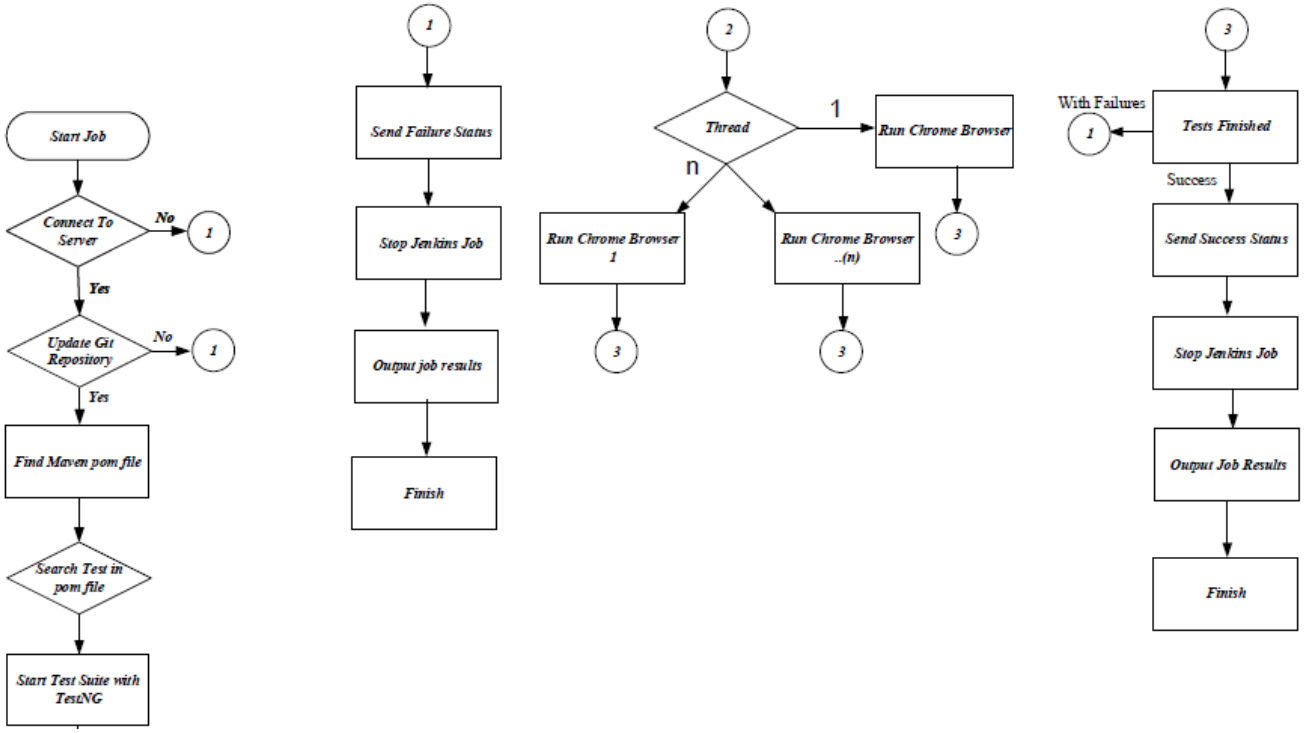
ДОДАТОК Г

Структура проекту у інтегрованому середовищі розробки



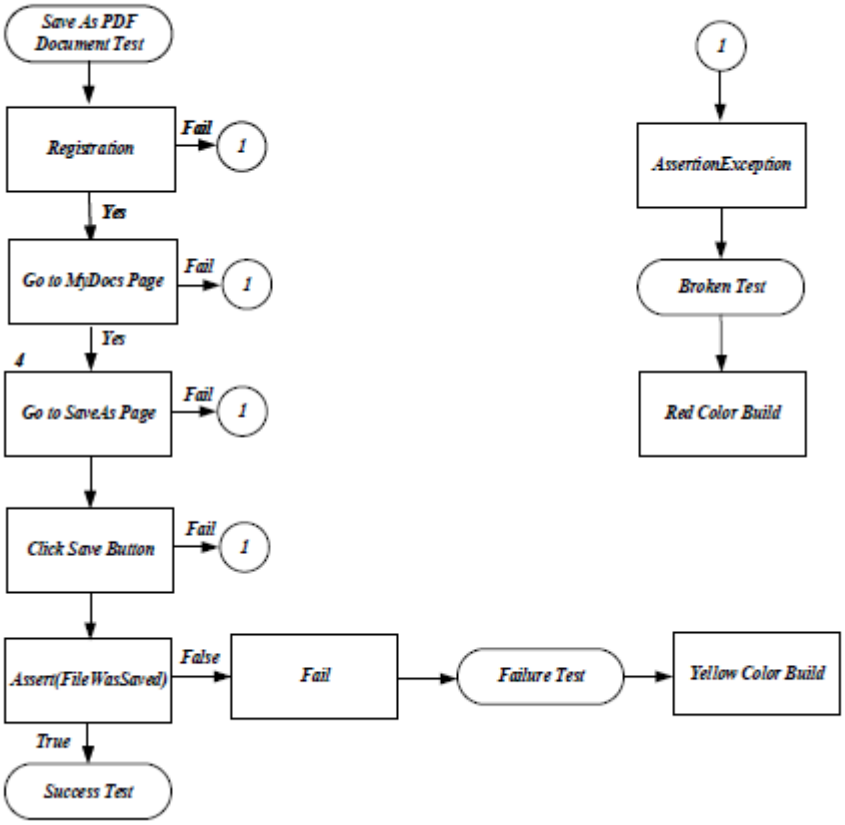
ДОДАТОК Д

Алгоритм роботи системи



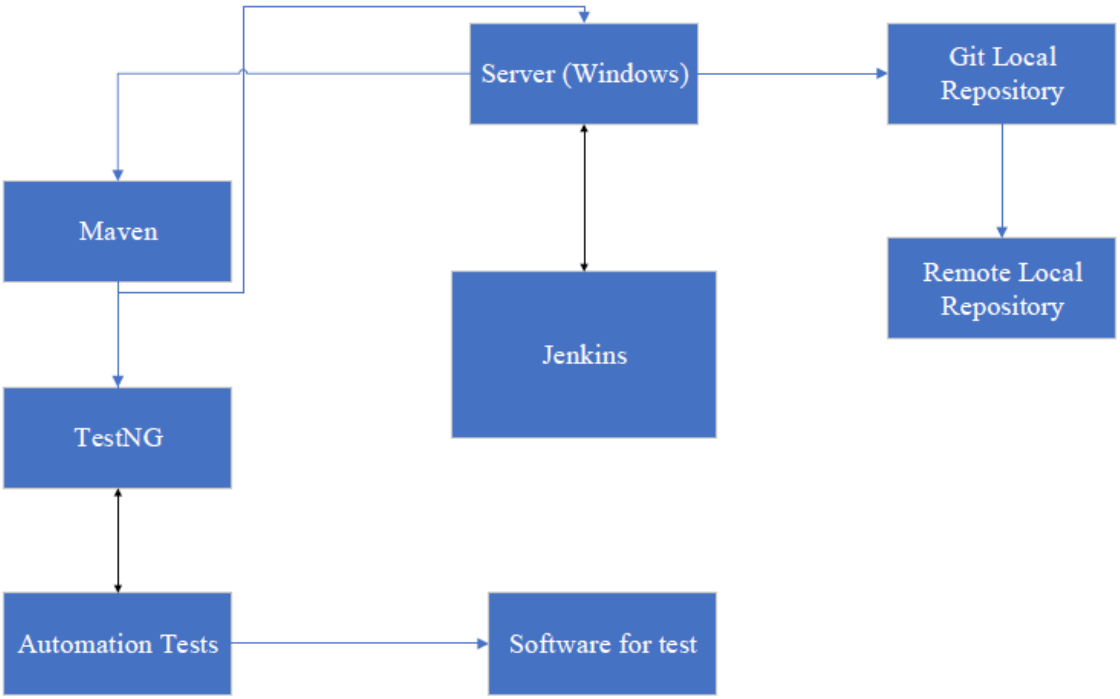
ДОДАТОК Е

Алгоритм роботи виявлення помилок при тесті



ДОДАТОК Ж

Структурна діаграма


Server (Windows)
Git Local Repository
Maven
Remote Local Repository
Jenkins
TestNG
Automation Tests
Software for test

ДОДАТОК З

Життєвий цикл програмного забезпечення використовуючи розроблену систему